

C# بعمق

خطوتك نحو الإتقان

01000011 00100011 00100000 01101001 01101110 00100000 01100100 01100101
01110000 01110100 01101000 00001010 01100010 01111001 00111010 00001010
01001000 01100001 01110011 01100001 01101110 00100000 01001101 00101110
00100000 01100001 01101100 00101101 01000110 01100001 01101000 01101100

بقلم

حسن الفحل

1442 – 1441   2020 – 2019

C# بعمق، خطوتك نحو الإتقان



إهداء

إلى مَن لولاهما ما كان هذا الكتاب

إلى أمّي وأبي

إهداء 2

إلى الغرباء

الذين يصلحون إذا فسد الناس!

إهداء 3

إلى أحد دكاترتي في الجامعة والذي قال لي "حاجتك برمجة وشغلات فاضية، ركز على دراستك"، أهديك كتابي الثاني وأترك لك دراستي لتركز عليها أنت

شكر وتقدير

لا أتخيل نفسي قد أنهيت هذا الكتاب ولم أشكر كلّ من ساهم في إتمامه ونجاحه، ولو بمعلومة أو كلمة أو نصيحة أو تشجيع أو حتى بدعم نفسي!!

أشكر بدايةً أساتذتي تركي وخالد، من أشعلوا أول شمعة في طريقي في عالم البرمجة. كما أدين لأستاذي م. شريف الشكر لما كوّنه في نفسي وشخصي في بداياتي في الجامعة، ولصديقي م. رامي لما كوّنه من فكري ومعرفتي بعد تخرجي.

{ عَلَّمَ الإِنْسَانَ مَا لَمْ يَعْلَم }

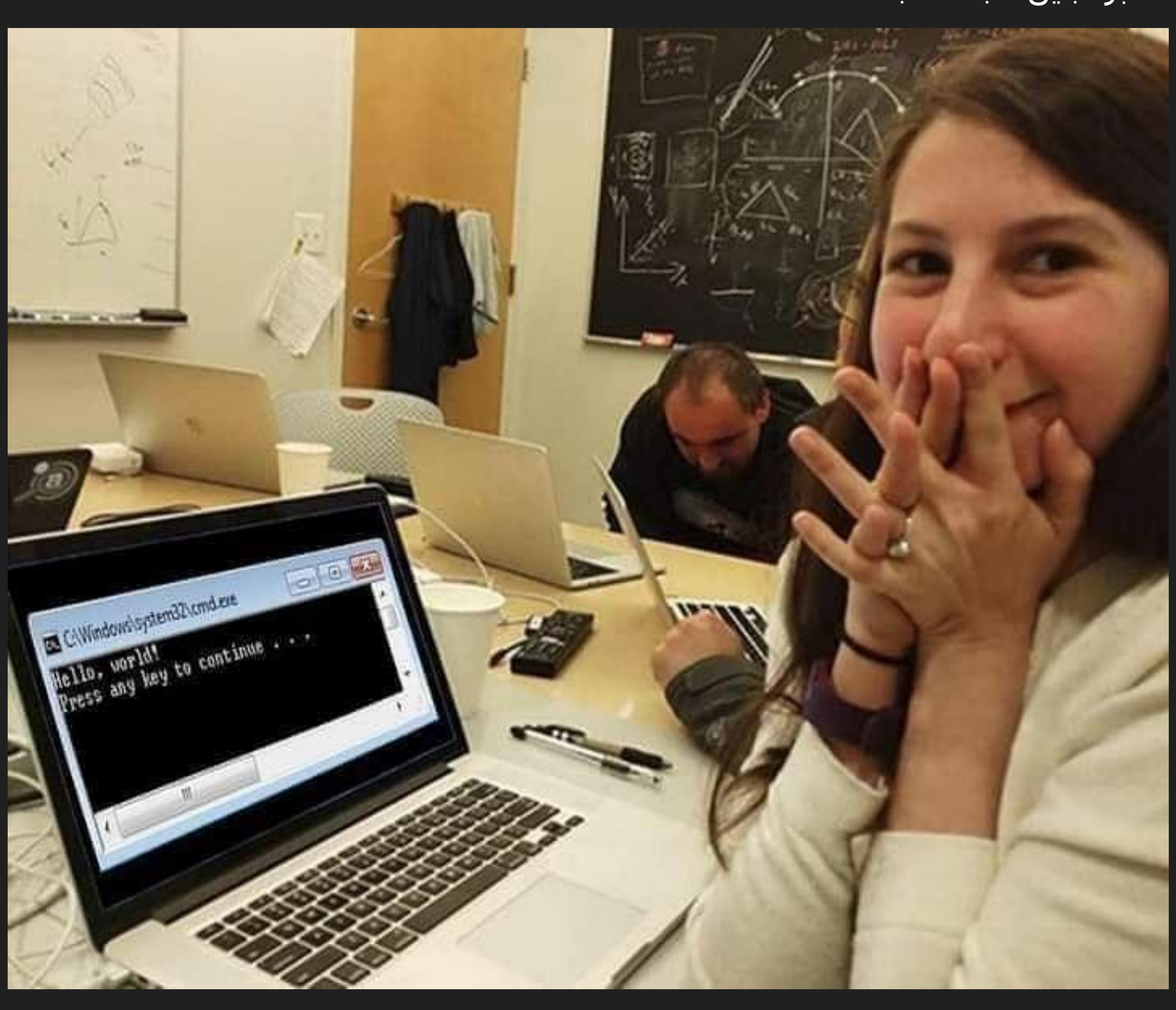
"مَنْ سَلَكَ طَرِيْقًا يَلْــتَـــمِـــسُ بِـــهِ عِلْمًا، سَهَّلَ اللهُ لَـــهُ بِـــهِ طَرِيْقًا إلى الجَنَّة"

رسول الله مُحَمَّد ﷺ

بداية المبرمجين الجدد، البرنامج الأول:

يتبع..

المبرمجين الجدد مجدّدًا:



المحتويات

الصفحة

الجزء الثالث **C# بعمق**	25
مقدمة	26
نبذة عن المؤلف	29
اخترعِ العجلة :)	29
أين نحن من الغرب؟	30
أين الغرب منا!	32
استثمر عقلك	34
كتاب برمجي آخر؟!	35
لتعلم التصميم، الكتب أم الفيديوهات؟	36
معضلة المصطلحات	37
لمن هذا الكتاب؟	38
خطة الكتاب	40
تنسيق الكتاب	42
مؤلّفات أخرى	44
التواصل مع المؤلف	44
الباب الأول **ما تحتاجه لتبدأ**	**46**
الفصل صفر – البرمجة كائنية التوجه OOP	**49**
لماذا OOP؟	50
الفئات والكائنات Classes And Objects	51

مبادئ OOP الأساسية 52
الكود النظيف في البرنامج السليم 58
- سمّ متغيرات برنامجك بأسماء ذات معنى 58
- طبق مبدأ المسؤولية الواحدة SRP 59
- التعليقات في الكود مثل الملح في الطعام، كثيره مثل غيابه 61
- طبق مبادئ SOLID 61

الفصل الأول – ما هي الـ UI وUX؟ 63

الألوان 67
النسق Layout 69
الخطوط 70
النصوص 71
اختصر خطوات الوصول لوظائف تطبيقك 71
القوائم الطويلة مملةٌ 72
اعتن بالتفاصيل 72
التصميم، للمستخدم أولًا وآخرًا 75
كلمة أخيرة 78

الفصل الثاني – أدوات تصميم تطبيقات ويندوز 81

الأدوات القياسية في ويندوز 82
- المؤشر Pointer 85
- زر الأوامر Button 85
- صندوق الاختيار CheckBox 86
- صندوق لائحة قابلة للاختيار CheckedListBox 86
- صندوق اللائحة المنسدلة ComboBox 86
- أداة انتقاء التاريخ والوقت DateTimePicker 88
- أداة العنوان Label 89
- أداة العنوان التشعبي LinkLabel 89

- صندوق اللائحة ListBox 91
- قائمة العرض ListView 91
- صندوق النص المُقَنَّع MaskedTextBox 94
- أيقونة الإشعارات NotifyIcon 96
- صندوق النص الرقمي NumiricUpDown 96
- صندوق الصورة PictureBox 96
- شريط التقدم ProgressBar 96
- زر الراديو RadioButton 96
- صندوق النص الغني RichTextBox 96
- صندوق النص TextBox 97
- أداة التلميح ToolTip 97
- شجرة العرض TreeView 97
- متصفح الويب WebBrowser 97
- لوحة ذات نسق تدفقي FlowLayoutPanel 97
- صندوق التجميع GroupBox 98
- لوحة Panel 98
- فصل الحاويات SplitContainer 98
- أداة التبويب TabControl 98
- العامل الخفي BackgroundWorker 99
- مزوّد الأخطاء ErrorProvider 101
- مقياس الأداء PerformanceCounter 105

إنشاء أدوات برمجيًّا 109

مصمم النموذج designer 111

- الإجراء InitializeComponent 113
- المنطقة Windows Form Designer generated code 114
- على أي أساس تُضاف أكواد إنشاء الأدوات في المصمم؟ 116

- التوثيق XML Documentation 119

مستعرض الكائنات Objects Browser 128

الفصل الثالث – تقنيات دعم البرامج 135

خدمات ويندوز Windows Services 135

- إنشاء خدمة ويندوز 137
- الفئة ServiceBase 139
- تغيير اسم خدمة ويندوز 139
- إضافة وظائف لخدمة ويندوز 140
- إنشاء مثبت لخدمة ويندوز 143
- إلغاء تثبيت خدمة ويندوز 148
- أتمتة عملية التثبيت وإلغاء التثبيت 148

جهاز إدارة ويندوز WMI 149

- تطبيق 1 – لمحة سريعة 149
- تطبيق 2 – مكتبة جاهزة 152
- تطبيق 3 – مكتبة جاهزة 2 157

الفصل الرابع – البنية التحتية للغات البرمجة 163

ما هي لغة البرمجة؟ 163

ما هي أفضل لغة برمجة؟ 164

ماذا لو انقرضت لغتي البرمجية؟! 165

كيف يمكن إنشاء لغة برمجة؟ 165

لكن لحظة.. لا تقلدا 168

كيف يمكن احتراف لغة برمجة ما دون الأساسيات؟ 170

الباب الثاني

الرسم والـ Graphics 171

الفصل الخامس – مدخل إلى الرسومات Graphics 175

مصطلحات ذات صلة 176

- الـ Interface 176
- الـ UI 178
- الـ UX 179
- الـ CUI 180
- الـ GUI 180
- الـ GDI 181
- الـ +GDI 182

برمجة الرسومات 182

- مجالات أسماء +GDI 182
- أدوات الرسم الأساسية 183

كائنات الرسم الأساسية 189

- النقطة Point 190
- الحجم Size 190
- المستطيل Rectangle 190
- الأقلام Pen objects 190
- فُرَش الرسم Brush objects 192
- كائن الرسومات Graphics 192
- كائنات أحداث الرسم PaintEventArgs 193
- الرسم دون الاعتماد على أحداث الرسم 195
- دقة الرسم 197
- كائن اللون 198
- رسم بعض الخطوط 198
- رسم بعض الأشكال وإملاءها 201
- رسم نجمة 204

رسم النصوص 206

- رسم نص داخل مستطيل 207
- رسم نص على محيط دائرة 208
- رسم نص محاذى نحو اليسار 210
- رسم نص محاذى نحو اليمين 211
- رسم نص محاذى نحو الوسط 212
- قياس النصوص 213

المناطق والمسارات 213

- اقتطاع الرسوم Clipping 215
- تقاطع مناطق الرسم 218
- إغلاق المسارات تلقائيًّا 220
- تظليل الرسم عند النقر على مسار ما 221
- التحقق من أن مؤشر الفأرة خارج أو داخل مسار ما 223

التلوين، بشكل متقدم 225

- التلوين المتدرج الخطي 225
- التلوين المتدرج المتشعب 236
- تهشير الأشكال 243
- تلبيس الأشكال بصورة bmp 245

التعامل مع الرسومات باستخدام الفأرة 248

- رسم خط عند الاستمرار بالنقر على منطقة الرسم 248

تحديث الرسم مع تغيير حجم منطقة الرسم 249

رسم التوابع الرياضية 251

- ضبط منطقة الرسم 251
- رسم المحاور الإحداثية 252
- رسم شبكة Grid 253
- رسم عنوان تفصيلي Legend 254
- آلية رسم المنحنيات 255

- رسم نقاط التقاطع مع المحاور الإحداثية 258
- رسم الخط البياني لمستقيم 259
- رسم الخط البياني لقطع مكافئ 261
- منحني بيزييه 264

الباب الثالث

أدوات المستخدم UserControl 269

الفصل السادس – مدخل إلى تصميم الأدوات 273

أدوات المستخدم عن كثب 273
- حقيقة الأدوات Controls 274
- المكونات Components 275
- صعوبات العمل مع أدوات المستخدم 275
- أساليب إنشاء أدوات المستخدم 275
- إنشاء أداة بإضافة ملف فئة Class للمشروع 276
- إنشاء أداة من خلال مشروع Windows Forms Control Library 278
- تجربة الأدوات واختبارها، برمجة الأدوات 279
- خصائص وطرق وأحداث الأدوات 281

تفاصيل ستجعل أدواتك أنيقة – احترافية 283
- وثّق ووصّف أدواتك 283
- اتّق هفوات المبرمجين بالمعدّدات 296
- اجمع الخصائص المتشابهة لأدواتك مع بعضها 297
- تخلّص من أعضاء الفئات غير المطلوبة 302
- ميّز الخصائص المهمة 305
- ظلّل أدواتك 311
- زوّد الأدوات بخصائص إضافية 312
- اعتمد على الفئة ControlPainter لرسم أدواتك من الصفر 312

- لا تتكلف! 313

البنية التحتية لمكتبة أدواتك الخاصة 314

- مجال الأسماء، وما فيه 315
- مواصفات وخصائص المشروع 316
- الفئات المجردة 317
- ماذا عن الواجهات Interfaces؟ 317
- أحداثك الخاصة Custom Events 324
- فئات الأدوات الأساس (الأم) BaseControls 325
- أدوات إدارة التصميم في وقت التنفيذ Run-Time 333

الفصل السابع – تصميم الأدوات 341

واجهات المكتبة 345

- الواجهة IEng27Animation 345
- الواجهة IEng27Border 346
- الواجهة IEng27Form 346
- الواجهة IEng27Radial 347

النماذج Forms 347

- النموذج Eng27Form 347
- النموذج FormShadow 350

أدوات قياسية مطورة 353

- ButtonFlat 353
- ButtonSquare 360
- TextBoxSingleLine 361

أدوات مدمجة 363

- Eng27UserControl 363
- TextBoxLabeled 365
- TextBoxLine 367

- TextBoxWatermark 369
- ListBoxReArrange 372
- ListBoxTransfer 379
- StepsProgress 383

أدوات ذات طبيعة متحركة AnimatedControls 389

- ButtonAnimated 389
- ButtonDoubleAnimated 395
- ButtonCircularAnimated 403

الأدوات المعدلة بالرسم OnPaint 408

- ButtonCircular 410
- CheckBoxCircular 413
- PictureBoxCircular 416
- ButtonImage 418
- ButtonGradiant 423
- PanelGradiant 425
- PanelRounded 427
- PanelShadow 429
- ToggleSwitch 431
- ToggleSwitchLabeled 434
- ProgressBarCircular 442
- TextBoxRounded 448
- TextBoxRoundedButton 454

أدوات الرسم Painting 457

- DigitPainter 457
- ColorBoard 460
- ChartPie 463

المكونات Components 473
- ColorTransition 473
- ControlRounding 477
- ControlShadow 479
- ControlScreenshot 484
- ControlExtensionProperty 486
- FileModel 490

أدوات مقترحة 501
ماذا عن WPF؟ 507

الفصل الثامن – الفئات الرياضية 509

الفئة System.Math 510
أساسيات رياضية 510
- الثوابت Constants 510
- الواحدات Units 512
- التعامل مع المعدّدات نصيًّا 512
- الفئة MathObject 513
- الفئة PhysicalObject 514
- العمليات الجبرية، برمجيًّا 515

فئات فيزيائية 516
- المسافة Distance 517
- الزمن Time 521
- السرعة Speed 525
- التسارع Acceleration 529

الأعداد العقدية 533
- الزاوية Angle 534
- العدد العقدي الديكارتي 536

- العدد العقدي القطبي (المثلثي) 543

الباب الرابع
فريموركات جاهزة **546**

الفصل التاسع – مدخل إلى تصميم النوافذ **549**

مكتبة Transitions 551
- أنواع النقلات 551
- تهيئة المشروع 552
- إنشاء النقلات 553
- حدث اكتمال النقلة TransitionCompletedEvent 553
- أمثلة 554

الفصل العاشر – منصات صغيرة **557**

منصة Metro 557
- بداية سريعة – صناديق الرسائل، وبعض الإعدادات 563
- نافذة تسجيل دخول بسيطة – صناديق النصوص 570
- قائمة مهام بسيطة 575

منصة XanderUI 578
- نافذة ملف شخصي بسيطة 579

منصة Guna.UI 586
- بداية سريعة – نافذة إعلانية صغيرة 588
- نافذة ملف شخصي صغيرة، منبثقة 593
- برنامج عرض خدمات 598
- لوحة تحكم بسيطة 604
- نافذة بسيطة لإدارة منتجات 608
- نافذة تسجيل دخول / إنشاء حساب 615
- نافذة تسجيل دخول متحركة 623

منصة Bunifu 628
- نافذة عرض خدمات 629
- نافذة ملف شخصي 637

الفصل الحادي عشر – منصة DevExpress 649

مدخل إلى منصة DevExpress 650
- المتطلبات الأساسية Prerequisites 650
- مستعرض نماذج ديف إكسبرس 651
- إعدادات مشروع DevExpress، ومظهره 658
- النموذج الافتراضي XtraForm 661
- نموذج الأشرطة RibbonForm 662

المشروع الأول، مشروع إدارة مهام 665
- القائمة الرئيسية MainMenu 666
- القائمة الرئيسية MainMenu، بعمق 677
- النوافذ متعددة المستندات MDI Forms 678
- تهيئة النموذج الأم في المشروع 678
- تهيئة النموذج الأم في المشروع، بعمق 682
- تهيئة النماذج الأبناء 691
- الأداة NavBarControl 694
- الأداة LayoutControl 697
- الأداة PopupGalleryEdit 702
- الأداة LayoutControl مجددًا، المجموعات والتبويبات 705
- التحكم بأدوات المشروع من خلال الكود 711
- الأمر جديد New 713
- الأمر حفظ Save 716
- الأمر حفظ الكل SaveAll 718
- الأمر فتح Open 719

- الأمر بحث Search 719
- تخصيص اختصارات للأدوات والعناصر 720
- تخصيص تلميحات للأدوات والعناصر 721
- عنصر البحث SearchItem 723
- تعدد اللغات Localizable 724
- تغيير سمة التطبيق Theme 726
- تفضيلات المستخدم واختياراته 727

المشروع الثاني، مدير متجر مبيعات وتخزين 727

- الأداة TileControl 731
- نافذة تسجيل الدخول 735
- نوافذ الخزنة والبنك 740
- نوافذ المخازن والصالات 745

الملحقات 752

الملحق أ – قواعد بيانات SQLite 753

الملحق بـ – إنشاء التقارير لقواعد بيانات SQLite 758

الملحق جـ – حماية قواعد بيانات SQLite 763

الملحق د – الفئات الوسيطة، نقل البيانات بين النوافذ، الإعدادات 764

الملحق هـ – كائنات البيانات الخاصة 765

فهرس المصطلحات Glossary 766


```
private void btnCalculation_Click(object sender, EventArgs e)
{
    Time t = new Time(15, TimeUnits_e.Minute);
    Distance d = new Distance(140);
    d.Unit = DistanceUnits_e.Kilometer;
    Speed s = d / t;
    Acceleration a = s / t;
    a.Precision = 3;
    MessageBox.Show(a.ConvertTo(AccelerationUnits_e.CentimeterPerSecond2).ToString());
}
```

17.3 cm/s2

OK

الجزء الثالث

C# بعمق

مقدمة

أما قبل: بسم الله، والصلاة والسلام على رسوله المصطفى، ورضوان الله تعالى على الصحابة الكرام، أبي بكر وعمر وعثمان وعلي، وحمزة والعباس، والحسن والحسين، وسائر الصحابة، والتابعين، وتابعيهم بإحسان إلى يوم الدين. والحمد لله الذي علم بالقلم، علم الإنسان مالم يعلم. وسبحانه إذ جعل هذه الدنيا دار ابتلاء؛ ليميز الخبيث من الطيب. وما أكرمه إذ جعل الثواب على النية قبل العمل، وعلى العمل بعد العزم عليه، ولم يطالبنا بالنتيجة، وإنما بالسعي لها. وأما بعد:

إن التقدم التقني والانفجار المعرفي وحوسبة معظم العلوم والأعمال، حتى النقلية منها، ومحدودية عمر الإنسان؛ يحتّم على المرء أن يتقن على الأقل لغة برمجية واحدة، أو

تقنية حاسوبية ما، فإن لم يكن ذلك لعمل أو مصلحة، كان للعلم والمعرفة، فالبرمجة منطق قبل كل شيء.

كما أن احتكار الغرب للتقنيات الرقمية – وهو إيجابي من عدة أبواب في هذا العصر مع فساد أممنا والأمم المجاورة – وإعطاءهم للعالم كل ما هو قديم ومستهلك، وسيطرتهم على مصادر المعرفة؛ يجب أن يكون حافزًا للإنسان – خصوصا ذلك الذي يعيش في الدول المتخلفة أو المستعمَرة أو الضعيفة عموما – لإغناء معرفته بالعلوم الحاسوبية، ليس من باب اللحاق بهم وعدم البقاء متخلفين عنهم [1]، وإنما من باب تحصيل الأدوات ووسائل المعرفة التي استفادت منها الأمم الأخرى، ليستفيد منها ويفيد، فإننا نؤجر بالنية، والعمل، وعمل من بعدنا بعملنا!

ثم إن المسلم كالغيث – كما يقول البشمهندس أيمن عبد الرحيم – أينما حل نفع، وعمله لله وحده، لا لقوم ولا لسمعة؛ فحري بنا بعد هذا أن نجدّ في طلب العلم ونشره، وألا نترك مَن طلب العلم للدنيا أن يسبقنا إليه وفي نشره ونحن طلبناه للآخرة!

ولهذا وذاك وغيره كان هذا الكتاب، فالحمد لله على توفيقه.

هذا الكتاب ليس كتابًا تعليميًّا للتصميم ولا للرسم، وإلا كنت سميته بذلك أو لمحت به في اسمه.. هذا الكتاب يتناول مواضيع متقدمة في البرمجة بلغة C#، بدءًا من البرمجة كائنية التوجه وحتى الرسم، مرورًا بتصميم الأدوات وتصميم النوافذ. كما أنه لا يفصّل لك هذه المواضيع، وإنما يسردها بأسلوب حواري يتناولها بشكل عام؛ لذلك فسترى كثيرًا من الروابط ضمن هوامش صفحات الكتاب لإحالتك لمصادر مختلفة تفصّل لك المواضيع أكثر، هذا فضلًا عن الكم الكبير من الأكواد غير المشروحة..

وبالحديث بشكل شخصي أكثر، وعلى اعتباري غير مختص من الناحية المعلوماتية، فإني أحذرك عند قراءة الكتاب، وأتمنى أن تبحث وتتحقق وتتبين من أي معلومة تقرؤها

[1] فهذا الإنسان الذي يحاول الوصول للمعرفة حتى لا يقال عنه متخلفا فقط، لن يهتم بها ولن يسعى إليها إذا رأى الغرب أنفسهم متخلفين ولا علم لهم، إذ إن الحافز قد زال!

في رحلتك مع هذا الكتاب، وأنصحك بذلك مع جميع ما تقرؤه، حتى مع تلك الكتب الي يقول أصحابها أن محتواه مضمون 100%، أو أنها أكثر الكتب مبيعًا أو قراءةً.

ومن ناحية أخرى، فإن ما دفعني لتأليف هذا الكتاب، ليس قلة المصادر والكتب في المجال الذي يتناوله الكتاب، ولا الحاجة لكتاب يتناول هذا المجال بشكل معين غير مسبوق؛ وإنما عادتي في توثيق كل ما أتعلمه، فتراني أكتب وأجمع وأنسق المعلومات التي تعلمتها من هنا وهناك، فإذا جمعت كمًّا كافيًا من المعلومات في موضوع ما، وضعته في فصل، وقسمته لفقرات. وهذا سر الوقت الكبير الذي أستغرقه لإنهاء الكتاب ونشره، فقد استغرق هذا الكتاب سنتين، وكتابي الأول كذلك، وسبب ذلك عمل ودراسة في اختصاصي، وانقطاع الكهرباء الذي وصل أحيانا إلى 23 ساعة في اليوم 🙂. وعلى ذمة برنامج الوورد فإن مجموع ساعات العمل على فصول هذا الكتاب حوالي 375 ساعة من أول كلمة كتبت في الكتاب وحتى وقت قريب من تحويله لـ PDF.

ومن نفس الناحية، فإن سوء المناهج التعليمية، ورداءة الكتب الجامعية، وضعف الكادر التدريسي، وقولبة العلم، وغياب المبادرات والمنظمات الأهلية المستقلة؛ هي أسباب غير مباشرة لوجود هذا الكتاب. وهذا ما لمسته في كثير من الدول بتواصلي مع عدد كبير من طلاب الجامعات من دول مختلفة. وأما ما يمكن أن يخفف من بؤس واقعنا، فلعله أن يكون اهتمام العقول النابغة والأقلام المبدعة بتوثيق العلوم حتى لو بالمجان (على شكل كتب أو دورات مجانية)، وإقامة الورش التدريبية، ونشر المعرفة وعدم احتكارها. والأهم من هذا وذاك، أن تكون نية العمل خالصة لله وحده، لنفع الأمة، والنهوض بها، حتى لا يصاب صاحب العمل بالإحباط إذا فشل عمله ولم ينتشر إذا كانت غايته الشهرة والمال، فله أجر النية والعمل، ولا شأن له بالنتيجة، هذا فضلًا عن أني لا أعلم عملًا كان لوجه الله إلا قُبل ونما، فنسأل الله القبول.

أواخر 2020، حلب

نبذة عن المؤلف

أنا حسن الفحل، مهندس إنتاج (خريج كلية الآداب الميكانيكية في جامعة حلب)، ومطور برمجيات. لست متخصصا في علوم الحاسوب، ولكني على اطلاع على بعضها، بما يكفي لإخراج بعض الأعمال التي قد ترقى للأعمال المعرفية.

كما أود أن أخبرك أنني أفتقر للقدرات الأدبية، لذلك فتوقّع وجود عشرات – لا بل مئات – الأخطاء على امتداد الكتاب، وأفضل مثال على ذلك كتابي السابق! لكن بالمقابل لا أعتقد أنك تتصفح هذا الكتاب لترفه عن نفسك، أو لتكتسب ثقافة أدبية، لذلك فإني أُعَوِّلُ على رغبتك بالتعلم واهتمامك بالعنب وتجاهلك للناطور، لذلك فسامحني.

اخترعِ العجلة :)

كثيرًا ما نسمع عبارة "لا تخترع العجلة مرة أخرى" خصوصًا عندما تسأل عن كيفية صناعة أشياء تمت صناعتها مسبقًا. ومقصد قائلي هذه العبارة أنه لا فائدة من اختراع أشياء اخترعها من سبقك، اخترع أشياء جديدة!

قد تبدو هذه العبارة حكيمة للهولة الأولى، ولكن نظرتي التشاؤمية للحياة تريني عكس ذلك.. فأنا أصلًا – كشاب عربي مفتقر للدعم والعلم والثقافة الكافية – لا يمكنني تخيل أشياء جديدة غير موجودة فضلًا عن اختراعها!! فبإعادة اختراعي للعجلة تُفتح أمامي آفاق كثيرة لم أكن لأصل إليها ما لم **أجرب** اختراع العجلة من جديد.

العجلة – ولا أقصد بها العجلة حرفيًّا – بسيطة بالنسبة لي ولك، لكن إن أردنا تقليدها لعجزنا، هذا ولم نفكر باختراع أشياء أحدث من العجلة أصلًا.

لا أقصد بهذا كله أن نخترع العجلة لمنافسة أصحاب العجلات المسيطرين على السوق أبدًا، وإنما أميل لاختراع العجلة لفهم مبدئها، لأحصل على الخبرة التي اكتسبها صانعوا العجلات من قبلي وبنوا على خبراتهم أشياء جديدة لم نكن لنتخيل وجودها يومًا!!

فلا بأس باختراع لغة برمجة جديدة أو نظام تشغيل أو منصات عمل مغايرة لمنصة DotNET أو حتى إنشاء مواقع تواصل اجتماعي عربية!! (هذا برمجيًّا، بنفس المنطق

قس ذلك على مناحي حياتك المختلفة). وأكرر، أقصد بـ "لا بأس باختراع" هذه الأشياء إن كان المقصد تعليميًّا، أما إن كان تجاريًّا فلا بأس به فقط في حال امتلكت الدعم الكافي لتنفيذ مشروعك ونشره وصيانته (لا تنس أن الشركات الكبرى ستشتري شركتك أو تهدمها إن كنت منافسًا حقيقيًا لها!!).

وستجد بين صفحات هذا الكتاب اختراعًا لعجلات كثيرة شائعة الوجود، منها البسيط ومنها المعقد، ولو بقيتُ بعقلية "لا يجب أن أخترع العجلة" لما كتبت صفحة من هذا الكتاب!! هذا فضلًا عن أن هناك "أشياء جديدة" ستجدها بين صفحات الكتاب أيضًا، والتي لو لم نخترع العجلات القديمة من جديد لما توصّلنا إليها!!

أين نحن من الغرب؟

لا شك أن كل مسلم في هذا العصر يمعن التفكير في واقعنا يستشعر ضعف الأمة، وهوانها، كونها مفتتة مشتتة، خصوصًا إذا ما نظر في التاريخ فرأى حضارة الإسلام منتشرة من أقصى الأرض لأقصاها؛ فيجد السفهاء مميزين، وأصحاب العقول مهمشين، والمفاهيم مفبركة، والمصطلحات فارغة مزخرفة، والضعيف لا سند له، والقوي لا تقوى عنده. فتجده يسأل: أين نحن من الغرب؟

لا شك أيضًا أن الدنيا للأصلح والأعدل والأحكم، عابدًا كان أو ملحدًا أو وثنيًّا حتى. ونحن في عصر تلونت فيه المفاهيم، فأصبح من الصعب تمييز الحق من الباطل، لما تشابه على الناس، فأصبح الحق حقوق، والباطل حريات، وأصبح من هب ودب يجعل نفسه صاحب الحق والمدافع عنه، فاخترع لك مفاهيمًا أجبرك على ارتداءها، وإلا قتلك باسم حقه المزعوم!

وقد أصبح واضحًا للعامي قبل العالِم سيادة الغرب على العالَم بأسره، وتحكمه في حضارات وثقافات الشعوب وتلاعبه بها، وأن الغرب هذا، كحالنا، وكحال غيرنا من الأمم، ومن قبلنا – ومن بعدنا إن لم يأت النيزك وينهي الوجود على هذا الكوكب البائس – له محاسن وله مساوئ، في ثقافته وحضارته، وربما تفوق علينا في كثير من الأشياء وربما تفوقنا عليه في غيرها (أقصد بـ الضمير "نا" في الجملة الأخير حالنا في هذا العصر

تحديدًا)، لذلك فمن الحكمة أن نأخذ أحسن ما لديهم وندعم به نقصنا واعوجاجنا، ونتعظ من أسوأ ما لديهم ونتذكر به استقامتنا؛ تماما كما فعل هو الغرب نفسه مع حضارتنا أول ظهوره وازدهاره.

ولكننا كعرب – ومن بحكمهم – ماذا فعلنا؟ انتقينا أكثر الأشياء عديمة الفائدة لدى الغرب وتبنيناها، وما يحزنك أننا قلدنا الغرب في الأمور عديمة القيمة بطريقة فاشلة! فحتى مسلسلاتنا وأفلامنا وأغانينا وبرامجنا التلفزيونية، الثقافية منها والإخبارية وغيرها، قلدناها بطريقة غبية، حتى الكتب والروايات انتقلت لها عدوى التغريب هذه، خصوصًا تلك المبنية على العامية والأحداث المستهلكة المتكررة التي تجدها في جميع الروايات! (مع أن المشكلة ليست في طرح الرواية بالعامية بحد ذاتها، وإنما بمبنى ومعنى محتوى الكتاب).

تفتح التلفاز لتشاهد مسلسلا يُدَّعى أنه لنقل صورة الواقع، فتجد أن أحداثه لا علاقة لها بالواقع، وأنها لا تدور حول مشاكل المجتمع ولا فيها ما يعالج قضاياه. حتى أن شخصيات هذا المسلسل لا تحوي شيئا من فقر المجتمع ولا حاجاته ولا آماله، رغم أن السواد الأعظم من المجتمع تحت خط الفقر! فما أكثر المسلسلات التي تدور حول "حياة الشباب السوري" تجد مجرياتها حول "كيف تعيش في دبي"، وحتى وإن كانت موجهة لقضايا المجتمع، فإن فيها من الانحياز والتزوير والكذب مافيها! هذا فضلًا عن روائح السيليكون التي تملأ الشاشة!!!

تنتقل لقناة الأخبار لتجد محتوى القناة يلهث حول التنبؤ بـ "أي قدم سيضع الرئيس الغربي الفلاني أولا، اليمنى أم اليسرى"، وحول تمجيد وتعظيم "طريقة عطسة الرئيس الغربي العلاني"، وحول "كيف سيغتصب بلادنا وينهب خيراتنا الرئيس الغربي الآخر"!!!

تنتقل لقناة ثقافية، لتجد مذيعيها يجهدون لنقل "رأي (خالد) في لون السيارات"، و"رأي (سمير) في رأي (خالد)"، في مشهد مليء بالديموقراطية وتبادل الآراء على الأشياء التافهة، مع أنك لا تعرف لا سمير ولا خالد ولا القضية المطروحة! هذا إذا كان محتوى "النقاش" يرقى لأن يقال عنه قضية أو نقاش حتى!

فإذا كان حالنا هكذا، فنحن نستحق الاستعباد والتبعية، فلا خير نقدمه لأنفسنا فضلًا عن أن نقدمه للعالم!

ولنخفف عن أنفسنا وطأة هذا السؤال المجحف "أين نحن من الغرب؟"، لا بد لنا أن نفكر بالعقلية التي فكر فيها أجدادنا المسلمون عندما فتحوا الأرض وملأوها علمًا وحضارة، عندما استوردوا علوم الأمم المجاورة ودرسوها وفهموها، وطوروا علومًا بناءًا عليها. وهذا أحد أسباب تأليف هذا الكتاب.

لكن مع ذلك، هذا لا يعني أن نلبس ثقافتهم، ولا أن نتكلم بلسانهم، ولا أن ننسى شخصيتنا! يجب علينا أن ننظر للعالم بثقافتنا نحن – ثقافة المسلمين – وأن نتعلم العلم بلغتنا نحن، وهذا لا يحدث في ظل "التعليم المجاني"، وقد صدق من أسماه "التجهيل باهظ الثمن"!

يقول البشمهندس: "تعلُّم اللغات الأجنبية واجب حضاري وشرعي، والتعلم باللغات الأجنبية خطيئة حضارية وشرعية"! ويقول أيضًا: "مفيش دولة محترمة في العالم بتعلم الناس بغير لغتها الأساسية".

وفي سياق غير متصل، تعال وسجّل منحة في جامعة "عربية" ما، ادخل لموقع هذه الجامعة، إذا رأيت كلمة عربية واحدة تعال وكلمني.

أين الغرب منا!

إذا كانت الفقرة السابقة قد سودت الدنيا بوجهك، فآمل أن تثلج قلبك هذه الفقرة، مع أني أتوقع أن القارئ على دراية بما كنت أتحدث عنه، إلا إن كان يعيش في دبي (ولا أقصدها حرفيًّا)!

نشر الأستاذ خالد السعداني يومًا:

وصلتني رسالتان اليوم من محاضر ومدون أجنبي خبير في البرمجة، يقول لي فيهما: أنه وجد كتابا لي يتحدث عن المبادئ الخمسة لتصميم البرمجيات وعلى الرغم من أن

الكتاب باللغة العربية الفصحى إلا أنه استشعر أهميته من خلال الأمثلة البرمجية التي وضعتها، وسألني هل يوجد نسخة مترجمة من الكتاب.

فأخبرته بأن الكتاب متوفر بالعربية فقط، فقال لي: في الأساس، أمثلة التعليمات البرمجية في كتابك مهمة، بل أكثر أهمية من كثير من الأمثلة التي تمكنت من العثور عليها، وسأتدبر أمري مع الكتاب حتى من دون وجود الترجمة.

قارنت بين حرصه واحترامه للكتاب، وبين أحد التعليقات التي قرأتها قبل مدة يسخر فيها أحد إخواننا العرب من عنوان الكتاب.

في نفس السياق، كنت قد ذكرت أحد كتبي في محاضرة من محاضراتي على الإنترنت، وفي التعليقات استفسر أحد الإخوة عن رابط الكتاب وأنه لم يجده، فأجابه أخ آخر بأن الكتاب غير موجود وأن الموجود فقط بضع مقالات على مدونة أكاديمية المبرمجين العرب ويبدو أن الأستاذ خالد اختلط عليه الأمر.

بعض التقصير يقع على شخصيا لأنني لم أنشر كتبي كما ينبغي، لكن ألا يليق بمن قرأ الكتاب واستفاد منه أن ينشره؟ ومتى كان الكاتب مطالبا بالقيام بنشر كتابه والتوسل للمكتبات والمنصات الإلكترونية؟ حوالي عشرين كتابا منذ عام ٢٠١٠ لم يأتني أي طلب من دار نشر لتعكف على طباعتهم، علما أنهم منشورون بالمجان ولا أنتظر منهم مقابلا.

ثم – سبحان الله – يأتي من يقول: المحتوى العربي فقير جدا.

دام لكم البشر والفرح. (انتهى منشوره)

فكتبت هذا التعليق في منشوره:

جزيت عنا كل الخير أستاذنا الكريم. لمست كلامك مرات عديدة، حتى إني – كشخص متوسط باللغة الإنكليزية، وغير مختص بهندسة الحواسيب وما يتعلق بالبرمجة؛ وبالتالي الكتب الأجنبية الأكاديمية صعبة علي – أقوم بالبحث عن الكتب العربية بالدرجة الأولى، وإن لم أجد، أذهب للمحتوى الأجنبي.

هذا وقد وجدت كثيرًا من الكتب العربية الجميلة ذات المحتوى القوي، صحيح أنها غير جيدة غالبا كتنسيق للكتاب، ولكنها كمعلومات جيدة، فغالبا كتب المحتوى العربي كتب لأناس هواة وليس لمختصين نشروا كتبهم بدور نشر محترمة.

لذلك، فلو حصل المحتوى العربي على الدعم لتأليف الكتب ونشرها، لحصلنا على كتب تضاهي وربما تغلب الكتب الأجنبية! [وكنت أقصد من الناحية الثقافية وليس العلمية].

كما أني وجدت أن بعض الكتب التي ألفها هواة عرب أفضل من الكتب التي تم ترجمتها، إذ إن الأولى تطغى عليها الثقافة العربية على اعتبار أن المؤلف يؤلف كتابه بثقافته، بينما الكتاب المترجم يتم سرده بثقافة غربية غريبة علينا (إلا إن كان المترجم قدير قادر على ترجمة الكتاب وتحويله من الثقافة الغربية للثقافة العربية، مع الحفاظ على المحتوى العلمي دون إفقاده قيمته) (انتهى تعليقي)

وعلى سيرة الكتب العربية، فإن المسلمين – منذ بدايات حضارتهم – حرصوا على تدوين العلم ونشره، فتناولوا شتى العلوم ووثقوا كل ما عرفوا، ولم يحتكروا، فقد عرفوا قيمة المعرفة، وكان جهدهم لله، فقطعوا مشارق الأرض ومغاربها بحثًا عن توثيق أو خبر. فأين الغرب من كل هذا؟ احتل البلدان، وقتل الناس واستعبدهم، وسرق الثروات، وصادر الآراء! فأين الغرب منا؟!

استثمر عقلك

هل ولدتَ وفي فمك ملعقة من ذهب؟ لا أظن ذلك. طيب هل لك من ترثه وتنقلب حياتك رأسًا على عقب وتصبح برجوازيا حقيرًا؟ احتمال ضعيف. هل تعيش في دولة ثرية وظروف عمل بلادك مناسبة ومدخولك مرتفع؟ أي هل تعيش في دبي؟ لو كان كذلك لما كنت تقرأ هذا الكتب. هل في بلدك ثروات ولا يوجد من ينهبها؟ error 404!

كثيرة هي البلاد الغنية بالثروات الباطنية، ولكن أغلب شعوب هذه البلاد فقراء. وكثيرة هي البلاد التي تفتقر بكثير من الثروات الطبيعية، الأساسية حتى! ولكن شعوبها من أعلى شعوب العالم دخلًا!

ولسوء الحظ، فإن هذه الحياة ليست لعبة جماعية إن لم تعجبك ظروف اللعبة غيرت السيرفر لتحصل على ظروف أفضل، فالسيرفر التالي هو يوم الحساب ولا سبيل لتجاوزه أو الانتقال إليه لمن ظن أن ما درسه للامتحان كاف قبل إنهاء هذا السيرفر.

في عصرنا الحالي، ومع تسارع العلوم وتقدم التقنيات، أصبح العمل الفردي أسهل، ومردوده مقبول في أسوأ الحالات، ورأس مالك خبرة وحاسوب، وبعض الإنترنت. كما أن التقنيات الحاسوبية التجارية – أو حتى تلك التي يمكنك برمجتها بنفسك – هي سبل كثيرة يمكنك سلوكها، ولا تحتاج إلا مجالًا تستثمر خبراتك فيه، وشيئًا من التسويق. وأما إذا استطعت تكوين فريق من مجالات متكاملة، فيا عيني على النجاح الذي يمكنك تحقيقه!

لا بل الأكثر من ذلك، حتى التعليم بات أسهل وأفضل، ففضلًا عن قنوات اليوتيوب الكثيرة التي يمكنك أن تتعلم منها مئات العلوم بشتى الثقافات واللغات، فإن هناك الكثير من المنصات التعليمية المجانية والتي تستضيف أرقى جامعات العالم، فلا تحتاج أكثر من جهاز حاسوب وإنترنت، وفي بعض الأحيان: هاتف محمول وإنترنت!

كتاب برمجي آخر؟!

دخلنا في 2021 وما زال أحدهم ينشر كتابًا جديدًا لتعليم البرمجة أو قناة على اليوتيوب لتعليم أساسيات لغة برمجة ما؟!

قد تكون على صواب إذا ما طرحت السؤال السابق، من زاوية ما، وقد يجانبك الصواب من زاوية أخرى. فلغات البرمجة – وعلوم الحاسوب – لا زالت علومًا قائمة وتقنيات مستخدمة، وخصوصًا لغات البرمجة على اعتبارها أبجديات علوم الحاسوب وأدوات التواصل بين الأشخاص والحواسيب.

صحيح أن كثيرًا من الكتب – حتى الأكاديمية منها – لا تقدّم محتوى برمجيًّا بمستوى مقبول، إلا أنك قد تجد بين الحين والآخر كتابًا فريدًا بأسلوبه وطرحه ومعلوماته، أسأل الله أن يجعل هذا الكتاب كذلك. ومن هذا الباب فإني أشجع على تأليف الكتب – أو حتى الكتيّبات والملخصات – ونشرها.

حتى أني لم أؤلف كتابي هذا – والكتاب الأول – بنية نشره أساسًا، وإنما غايتي كانت تجميع أفكاري ومعلوماتي في ملخص صغير، وسرعان ما تحول هذا الملخص إلى كتاب. ولعل سبب هذا أني لا يمكنني تعلم شيء دون تلخيصه، ولسوء خطي، الكتابة على برنامج الوورد هو سبيلي الوحيد لتلخيص معلوماتي، فرب ضارة نافعة!

وبشكل مشابه للكتب، فإن قنوات اليوتيوب والدورات البرمجية التي تجدها عليه يغلب عليها المحتوى الرديء، وتكاد تكون الدورات – أو القنوات – الجيدة معدودة!

ثم إن هذا الكتاب غير موجه لتعلم أساسيات لغة C#، بنسخها المختلفة، الحديثة خصوصًا، أو العمل على فيجوال ستوديو، أو مع منصة DotNET، بنسخها الحديثة خصوصًا؛ وإنما يناقش القارئ بمفاهيم برمجية يمكنه تطبيقها على أي لغة برمجة وبأي نسخة منها، فلا تتوقع تعلم لغة ما بانتهاءك من قراءة الكتاب.

وبأسلوب دراما وسائل التواصل العربية: لست ساحرًا ولا دجال، ولكنك غالبًا إما قد تابعت دورة خالد السعداني أو حسونة أكاديمي.

لتعلم التصميم، الكتب أم الفيديوهات؟؟

لنطرح السؤال بشكل مختلف: هل تعلُّم التصميم من خلال الكتب مجدٍ ومفيد؟ فالتصميم – سواءً أكان للصور أم للفيديوهات أم للنوافذ والأدوات البرمجية – يحتاج إلى معلّمٍ فوق رأسك يرشدك خطوة خطوة، وبإمكانك اعتبار اليوتيوب معلّما في وقتنا، خصوصًا في المواضيع التقنية. والإجابة على هذا السؤال برأيي أنه ممكن بالنسبة للأساسيات فقط. يمكن احتراف البرمجة من خلال الكتب أما التصميم فلا أعتقد ذلك، إذ إن متغيرات عملية التصميم كثيرة وقد تختلف كلّ ثانية أو اثنتين (خصوصًا عند التعامل مع طبقاتٍ عدة Layers)، ومن الصعب جمع صور بعدد الأمور التي تتغير أثناء تطبيق أفكار الدروس في صفحات كتاب واحد، أما في البرمجة فكثير من الأحيان لا تحتاج في الدرس الواحد أكثر من صورة أثناء تصميم البرنامج، وأخرى أثناء تنفيذه، والكود..

بالمقابل فإنني لاحظت أن الساحة العربية تكاد تكون خالية من الكتب التي تتحدث عن التصميم في البرمجة (أو ربما عمليات بحثي كانت قليلة)، كما أنني جمعت الكثير من

الأفكار من قنوات يوتيوب البرمجة العربية والأجنبية – والأجنبية محتواها أكثر وأوسع وأغنى – سواءً في التصميم أم في البرمجة وارتأيت أن أضعها في كتاب واحد، على الأقل أن أذكرها كأفكار وأدعمها ببعض الأشكال والأكواد، آملًا أن يحقق الكتاب الغاية التي وضع لأجلها.

معضلة المصطلحات

عند ترجمة محتوى علمي – أو غيره – من لغة لغيرها، فإن المصطلحات قد تختلف من كتاب لآخر، حتى لو كان المرجع الأصل نفسه. وهذا يعود للمترجم نفسه. هذه المعضلة تظهر بشكل أوضح عند التعامل مع علوم دخيلة على اللغة، فحتى لو لم تكن تترجم من كتاب بعينه وإنما تنتقي معلوماتك من عدة مراجع (بما فيها المواقع)، وتضع خبرتك وأبحاثك المختلفة في عمل ما، فإنك ستقع بمعضلة المصطلحات.

ترجمت بعض المصطلحات وفق ما هو متعارف عليه في الأوساط التي أتفاعل معها، لذلك فمن المحتمل أن تقع بالتباس أو سوء فهم عند تعرضك لبعض المصطلحات ضمن الكتاب، خصوصًا إذا كنت على اطلاع على ترجمات أخرى، وهذا ما دفعني لأن أُرفق المصطلح الإنكليزي مع المصطلح العربي في كل مرة يأتي فيها. كما أني أنشأت فهرسًا للمصطلحات Glossary في نهاية الكتاب، فيه جميع المصطلحات البرمجية الواردة ضمن الكتاب، مع شرح موجز لها، مرتبًا أبجديًّا مرة بالعربية ومرة بالإنكليزية.

حاولت الابتعاد قدر الإمكان عن الترجمة الحرفية للمصطلحات، والاعتماد على الترجمة المتعلقة بالسياق، حتى لو لم تكن الترجمة الحرفية تعني ذلك أو كان هناك مصطلح عربي آخر يدل على المعنى الحرفي للمصطلح الأجنبي بشكل أفضل. فمثلًا ترجمت كلمة Class على أنها "فئة"، ولو أن هناك الكثير من المراجع العربية – الأكاديمية منها خصوصًا – التي تترجمها على أنها "صف"، إذ إن معناها – برمجيًّا – أقرب للفئات منها للصفوف! وبالمثل، كلمة Object ترجمتها على أنها "كائن" عوضًا عن الترجمة الركيكة الشائعة في بعض المراجع "غرض".

كما أني ترجمت كلمة Control على أنها "أداة"، مع أن ترجمتها الحرفية لا علاقة لها بالأدوات لا من قريب ولا من بعيد، إلا أن معناها يشير إلى ذلك. ومن يتأمل منتجات مايكروسوفت يستشعر ذلك عندما يجد الأشياء المسماة بـ Controls مجمعة في صندوق يسمى ToolBox.

لمن هذا الكتاب؟

يتناول الكتاب مواضيع متقدمة في #C، كتصميم النوافذ والأدوات البرمجية، أو مشاريع تطبيقية، أو استخدام منصات عمل Frameworks جاهزة، وكل ذلك يحتاج إلى خبرة في OOP، لذلك فقد يتعذر للمبتدئين استيعاب المحتوى بالمستوى المطلوب. فهو موجّه للمتوسطين في #C، والمتقنين بتقدير جيد ربما. وحتى المبتدئين الذين عندهم اطلاع ولو بسيط على مبادئ البرمجة وأدواتها (بنى الشرط وحلقات التكرار وغيرها) وقد بدأوا لتوّهم في تعلم مبادئ البرمجة كائنية التوجه؛ يناسبهم هذا الكتاب، خصوصًا فصول الباب الأول منه.

ولكن عمومًا، القارئ نفسه هو من يحدد هل الكتاب الفلاني موجه له أو لا، فالقراء لا يخرجون عن ثلاثة أنواع:

1- من يقرأ ولا يفهم شيئا مما يقرأ؛ وبالتالي فالكتاب الذي يقرؤه غير موجه له لأنه لم يصل لمستوى محتوى الكتاب.
2- من يقرأ ويفهم ويعرف ويستوعب كل ما يقرأ؛ فهو أيضًا لا يناسبه كتابه هذا لأن مستواه أعلى من مستوى محتوى الكتاب بمراحل (كأن تقرأ كتابًا للصف الابتدائي).
3- من يقرأ ويعرف بعض الأشياء ويجهل بعضها؛ فهذا يناسبه الكتاب وهو بحاجة إليه.

وعليه، فإذا كانت أصول المعرفة البرمجية لديك كافية لفهم الطرق Methods والفئات Classes ومفاهيم البرمجة كائنية التوجه OOP، ولكن لا تملك تطبيقات برمجية واقعية لهذه المفاهيم؛ فإن هذا الكتاب مناسب لك بتقدير ممتاز. ومع ذلك فإني سأتناول بين

الحين والآخر هذه المفاهيم بشيء من الإيجاز لإنشاء أرضية معرفية مشتركة بيني وبين القارئ، خصوصًا في الفصول الأولى.

أما ماهية المفاهيم التي تحتاجها من مفاهيم OOP للاستفادة من هذا الكتاب بأكبر قدر ممكن، فهي لا تتجاوز ما يحويه المحتوى المرئي والمقروء العربي الذي يشرح OOP، أو حتى أقل منها أحيانًا.

وعلى سيرة المحتوى العربي، هل لاحظت – عزيزي القارئ – أنه رديء وموجّه للمبتدئين فقط ولا يرقى لمستوى إنتاج برامج محترمة ومفيدة؟؟ جميع كتبنا ودوراتنا المرئية تتحدث عن المتغيرات وجمل التحكم والتكرار وأساليب تجميع البيانات كالمصفوفات وأساسيات OOP فقط، وربما يعود ذلك إلى أن المؤلفين هم إما طلاب جامعات لديهم بحث أو مشروع قاموا بنشره تحت اسم كتاب برمجي، أو أناس هواة – أمثالي – رتّبوا أفكارهم في فصول متلاحقة ليشكّلوا كتابًا – أو دورة على اليوتيوب – باسم برّاق، لتتفاجأ بمعلومات سطحية جدًا، مع بعض الاستثناءات أحيانًا.

وبالحديث عن الكتب بالتحديد، فهي لا ترقى لاعتمادها كمراجع، فضلًا عن التعليم بها، خصوصًا المنشورة بدور نشر منها، لما فيها من ركاكة في الأسلوب (مع بعض الاستثناءات أحيانًا). بينما الكتب الأجنبية – التي قد يصل عدد صفحاتها إلى الآلاف – فتجدها جيدة المحتوى والتنسيق، وربما يعود ذلك للدعم الذي حصل عليه مؤلفو هذه الكتب في حياتهم، سواءً أثناء دراستهم أو في عملهم، وأثناء تأليفهم للكتب.

إذا لم تكن قد قرأت كتابي الأول فأنصحك بقراءته قبل قراءة هذا الكتاب، لأن الكثير من المعلومات الواردة في هذا الكتاب مرتبطة بالكتاب الأول، إذ إن هذا الكتاب هو الجزء الثالث لكتابي الأول. (ومع هذا فإنني سأضع بعض الفصول تحوي شرحًا لكثير من الأفكار التي سترد لاحقًا في هذا الكتاب، سواءًا أذُكرت في الكتاب الأول أم لم تذكر).

خطة الكتاب

على عكس ما قد تتوقع، ليست نيتي أن يكون الكتاب تعليميًّا، فالأسلوب الذي أخرجت به هذا الكتاب أسلوب حواري قابل للنقد والقبول والرفض، فجميع معلومات وفقرات وفصول وأبواب هذا الكتاب محتواها ليس إلا آراء شخصية لي مدعومة ببعض الأسس العلمية، يمكنك رفضها أو قبولها، وبقليل من البحث يمكنك إيجاد وسائل وتقنيات أفضل من بعض ما ستجده بين صفحات هذا الكتاب. حتى المعلومات التي سأقدمها على أنها قواعد أو أسس برمجية الغاية منها ليست تعليمية، وإنما لبناء أرضية معرفية مشتركة تمكننا من الانطلاق في الفصول والفقرات اللاحقة.

يضم الكتاب أربعة أبواب، الباب الأول موجه للمبتدئين، أما الأبواب الأخرى فهي موجهة للمتوسطين في لغة سي شارب.

يحوي الباب الأول **ما تحتاجه لتبدأ** خمسة فصول، ويحكي للقارئ بعض مبادئ البرمجة كائنية التوجه ومدخل إلى التصميم في البرمجة وبعض التقنيات المفيدة لدعم برامجه ونبذة عن لغات البرمجة.

الفصل الصفر **البرمجة كائنية التوجه OOP** يعطيك أرضية جيدة لتؤسس عليها ما ستقرؤه في الفصول التالية، وبعض فقراته لا تحتاج خبرات برمجية كبيرة لفهمها، فهو يناقش بعض أساسيات ومفاهيم البرمجة كائنية التوجه، والأدوات المستخدمة في تصميم مشاريع الواجهات على اعتبار أن بعض الفصول التالية تناقش إنشاء الأدوات من الصفر، وبعضَ المواضيع الأخرى.

الفصل الأول **ما هي الـ UI وUX؟** يستعرض بعض الاعتبارات التصميمية للحصول على منتجات برمجية جيدة، ومقبولة للعين والعقل.

الفصل الثاني **أدوات تصميم تطبيقات ويندوز** يسرد أكثر الأدوات شيوعًا في تطبيقات ويندوز، ويناقش بعض الأدوات التي يقدمها الفيجوال ستوديو لدعم المشاريع.

الفصل الثالث **تقنيات دعم البرامج** يشرح كيفية الاستفادة من بعض التقنيات الجاهزة الموجودة في ويندوز، كخدمات ويندوز وتقنية WMI، وقد سبق وتناولنا الرجستري وموجه

الأوامر في كتابنا الأول، وهي مواضيع هامة، حتى لو أنها قديمة، فكما يقول المثل: هذا الويندوز من ذاك الدوس! فليُراجَع.

الفصل الرابع **البنية التحتية للغات البرمجة** يناقش بعض الأسئلة الأكثر شيوعًا عن لغات البرمجة.

يضم الباب الثاني **الرسم والـ Graphics** فصلًا واحدًا هو **مدخل إلى الرسومات Graphics** والذي يستعرض أساسيات الرسم في البرمجة ويفصّل بعض تطبيقات الرسم، ومنها رسم التوابع الرياضية.

في الباب الثالث **أدوات المستخدم UserControl** يوجد ثلاثة فصول، تبدأ بلمحة عن المفاهيم البرمجية والاعتبارات التصميمية التي يجب أخذها بعين الاعتبار عند تصميم وبرمجة الأدوات، وتنتهي بتصميم أدوات Controls حقيقية منها المعروف ومنها الغريب.

الفصل السادس **مدخل إلى تصميم الأدوات** يستعرض اعتبارات تصميمية مختلفة، ويمهد للفصل السابع.

الفصل السابع **تصميم الأدوات** يبدأ ببعض الفئات الأساسية لتنظيم الأدوات معًا في مكتبة واحدة، ثم يفصّل كثيرًا من الأدوات الشائعة وغير الشائعة.

الفصل الثامن **الفئات الرياضية** يناقش كيفية تحويل الأفكار الرياضية إلى برمجية وترجمتها إلى فئات للحصول على تطبيقات رياضية مفيدة.

ويحوي الباب الرابع **فريموركات جاهزة** ثلاثة فصول، تعنى بتصميم النوافذ.

الفصل التاسع **مدخل إلى تصميم النوافذ** يناقش الأسس التي التي تبنى عليها النوافذ، وذلك باقتضاب شديد.

الفصل العاشر **منصات صغيرة** يتناول تصاميم جاهزة لنوافذ مصممة على بعض المنصات الصغيرة، كمنصة ميترو وBunifu وغيرها.

الفصل الحادي عشر **منصة DevExpress** يفصّل أهم المفاهيم عند التصميم باستخدام أدوات المنصة ويشرح مشروعين تطبيقيين، وكثيرًا من الأدوات الجاهزة في المنصة.

وقد حسبت حساب الرياضيين في هذا الكتاب، فوضعت لهم فصلين أحدهما لرسم التوابع الرياضية والآخر لإنشاء الفئات الرياضية وتمثيلها برمجيًّا.

كما أني وضعت ضمن ملحقات الكتاب بعض المواضيع التي يمكن أن تفيد المبرمجين في المشاريع الصغيرة، كقواعد بيانات SQLite.

ما يميز هذا الكتاب هو أن فصوله متسلسلة وغير متسلسلة في نفس الوقت، يمكنك قراءته من بدايته حتى نهايته لتحظى بفائدة تسلسل الأفكار كحلقات مترابطة متعلقة ببعضها، تمامًا ككتابي الأول. كما يمكنك قراءة فقرات أو فصول محددة من الكتاب بغض النظر عن الفقرات أو الفصول الأخرى، فلست ملزمًا أن تقرأ الكتاب بترتيبه لتحصل على الفائدة المرجوة منه، بالمقابل إذا أردت قراءته كاملًا فقد تم ترتيبه لتحصل على ترابط في الأفكار بحيث يتعلق آخره بأوله.

تنسيق الكتاب

لأخرج عن المألوف، جعلت لون خلفية صفحات الكتاب اللون الرمادي (32; 32; 32)، حتى يعطي الكتاب إشارة إلى بيئة البرمجة بالفيجوال ستوديو – أو غيرها من بيئات البرمجة والتصميم – والتي يسعى كثير من المبرمجين إلى جعلها بالوضع الليلي إما تقليدًا لجزء كبير من المبرمجين أو لارتياحهم لها.

أما بالنسبة لحجم الخط فهو 14 كالعادة، و20 بالخط العريض للعناوين العريضة و16 بالخط العريض للعناوين الفرعية، و9.5 للأكواد. وبالنسبة لشكل الخط فاعتمدت الخط Tahoma لجميع محتوى الكتاب إلا الأكواد فهي Consolas، كما في الفيجوال ستوديو.

اعتمدت في هذا الكتاب أسلوب الفقرات والفقرات الفرعية، بحيث تضم الفقراتُ الرئيسية – ذوات العناوين العريضة – الفقراتِ الفرعية، ومن المحتمل عدم وجود فقرات فرعية ضمن فقرة رئيسية ما. وكلها ضمن فصول، ضمن أبواب.

في كتابي السابق كانت لدي نية بترقيم الصور والأكواد، الأمر الذي لم أتمكن منه لضيق الوقت وقتها، أما في هذا الكتاب فقد ركزت على طريقة كتابة الأكواد والملاحظات والقواعد، كما أضفت مصطلح التلميحات إلى مفردات الكتاب، ولا زالت لدي نية بترقيم الصور والأكواد، الأمر الذي لم أتمكن منه أيضًا لضيق الوقت 🙂.

وضعت الأكواد في صندوق من الشكل:



والملاحظات:



والتلميحات:



أما القواعد والصيغ البرمجية:



مؤلَّفات أخرى



كتاب #C من البداية حتى الإتقان[1]، عدد صفحاته 510، يتناول الكتاب أساسيات لغة سي شارب وبعض المواضيع المتقدمة فيها، كقواعد البيانات والرجستري وموجه الأوامر، وبعض المشاريع التطبيقية.

التواصل مع المؤلف

للحصول على أي دعم أو استشارة، أو لاقتراح تعديلات على الكتاب أو الأعمال القادمة – إن قدر الله لي ذلك – أو لاقتراح أعمال، بإمكانك التواصل معي على بريد إحدى صفحاتي الخاصة بالبرمجة:

صفحة Eng27 – Programming على فيسبوك[2]، أو على تلغرام[3].

صفحة Eng27 – SourceCode على فيسبوك[4] أو على تلغرام[5].

أو من خلال التلغرام على المعرّف Eng27@، أو من خلال الرقم 627 796 954 963+، على واتساب أو تلغرام.

كنت قد أنشأت بوت على تلغرام باسم Eng27، والذي كنت قد وضعت فيه بعض أعمالي، ومجموعة من المراجع المفيدة، إلا أنه تعطل بسبب "توقف خدمة صناعة البوتات التي

---

[1] تحميل كتاب #C من البداية حتى الإتقان https://eng27-pro.blogspot.com/2020/12/books.pro1.html.
[2] صفحة Eng27 – Programming على فيسبوك https://www.facebook.com/Eng27Programming/.
[3] قناة Eng27 – Progamming على تلغرام https://t.me/Eng27Channel.
[4] صفحة Eng27 – SourceCode على فيسبوك https://www.facebook.com/Eng27SourceCode/.
[5] قناة Eng27 – SourceCode على تلغرام https://t.me/eng27sourcecode.

استخدمتها، على اعتبارها خدمة خاصة وليست رسمية ضمن تلغرام" وفق ادعاء أحد أعضاء فريق الدعم في تلغرام. وبعدها (في الواقع بعدها بفترة طويلة)، أنشأت مدونة، لنفس السبب السابق، آملًا ألّا تتوقف خدمة صناعة المدونات التابعة لشركة غوغل 😔.
يمكنك الاطلاع على هذه المدونة من هنا [1].

كما يمكنك الاشتراك بقناتي على يوتيوب [2] التي لم أنشر أي فيديو تعليمي فيها حتى الآن 🙂.

[1] مدونتي https://eng27-pro.blogspot.com/.
[2] قناتي على اليوتيوب https://www.youtube.com/channel/UCJirbWhb23DcZhsZ1pWPt8g.



الباب الأول

ما تحتاجه لتبدأ

يُفترض منك كقارئ لهذا الكتاب أن تكون ذو خبرة جيدة في #C بحيث تكون الأساسيات لديك من البديهيات، تمامًا كالأحرف والكلمات والجمل في اللغة العربية، أو الإنكليزية إن كنت تعيش في دبي🙂. وما ستقرؤه في صفحات هذا الكتاب سيعطيك أساليب متقدمة في البرمجة كأساليب الإنشاء والمواضيع البلاغية في اللغة العربية مثلًا، لذلك فلا تتوقع وجود قواعد أو شروحات عن أساسيات البرمجة.

من جهة أخرى، أرى أن وجود فصول تمهيدية أمر حسن مفيد، لذلك فقد ابتدأت الكتاب بفصول تبسط لك بعضًا مما قد يمر عليك خلال مشوارك مع هذا الكتاب، لكن لا تتوقع مني مثلًا أن أشرح لك ماذا تعني namespace أو كيف يتم استخدام مكتبة DLL كمرجع من مراجع البرنامج باستخدام الكلمة using أو كيف تقوم بالتصريح عن مصفوفة 😃، ويُتوقع منك أن تكون قد سمعت بجميع هذه الأفكار ولو كعناوين، وإلا فلا أنصحك بإكمال القراءة، فلا أرغب بناقد إضافي عن سوء فهم.

الفصل صفر – البرمجة كائنية التوجه OOP

يفترض مبدأ OOP أن كل شيء في البرمجة عبارة عن كائنات، وفيه يتم تصنيف كل شيء وفق أصناف معينة Classes، حيث يتميز كل كائن بمجموعة من الأفعال تسمى طرق أو توابع [1] Methods، ويتصف بمجموعة من الصفات Properties، وتمر عليه مجموعة من الأحداث Events، وذلك كله تبْعًا للفئة التي اشتُقَّ – أو استنسخَ Instanced – منها.

باختصار، عند تطبيقك لمبادئ البرمجة كائنية التوجه OOP في برامجك فإنك ستضفي روح الحياة عليها، ستجعل قراءتها أسهل، وفهمَها وتخيل وظيفتها.. من اسمها، هي

[1] يعتبر بعض المبرمجين أن ما نسميه بالطرق Methods هو توابع سواءًا أكانت تعيد قيمة أم لا تعيد، في حين أن غيرهم يقسّم هذا المفهوم إلى توابع Func تعيد قيمة وإجراءات Sub لا تعيد قيم، والأخير برأيي هو الأصح، ووفق ذلك سأستخدام مفهوم الطرق على امتداد فصول الكتاب كترجمة لـ Methods، والتي يمكن أن تكون توابعَ أو إجراءات.

أسلوب برمجي يعتبر أن المكونات البرمجية عبارة عن كائنات. عوضًا عن أن تنشئ كودًا يقوم بقراءة الملفات مثلا، وكودًا آخر يقوم بالتعديل عليها، وآخر يقوم بالبحث فيها عن معلومات معينة، و.. و.. (طبعا هذه كلها يمكن القيام بها بالبرمجة الإجرائية [1])، يمكنك عوضًا عن ذلك كله إنشاء ما يسمى بـ كائن، يقوم بكل هذه الامور وهو عبارة عن كتلة Block واحدة، عوضًا عن أن يكون مجموعة من الكتل.

هذا الأسلوب سيجعل البرمجة أسهل، سيجعل العمل ضمن فريق أفضل (تخيل العمل مع عدد من المبرمجين على مشروع معين، عند حدوث خطأ ستتراشقون التهم "من كتب هذا السطر!"، "من حذف الفاصلة المنقوطة من ذاك السطر!"، "من نسي تكرار الكود الفلاني بعد الكود الفلاني!!"، إلخ إلخ)، سيجعل إمكانية استخدام الكود في أكثر من برنامج أسهل (الكود سيحفظ ضمن ملف مصدري، غالبًا ملف DLL، ببروتوكول معين تفهمه جميع البرامج التي ستتعامل مع هذا الملف الحاوي على الكود)، كما ستجعل إمكانية تعديل الكود أسهل وأبسط وأكوس [2]. لذلك فخذ مني نصيحة، بعد انتهاءك من مقدمة هذه الفقرة، انتقل للفقرة التالية ولا تلتفت لأسئلة سطحية تشككك بـ OOP.

قد تتساءل: وهل البرامج المكتوبة بلغة السي – مثلا – لا توجد فيها روح؟؟ وسأجيبك: لا، لا توجد فيها الروح الواقعية. الأكواد التي في لغة سي فيها روح برمجية فقط شأنها شأن بقية لغات البرمجة، أما في اللغات التي تطبق OOP فإنك لو فهمتها ستشعر بالواقعية في أكوادك.

لن أتناول كل ما يخص OOP لأن هذا لا يتعلق بتوجّه الكتاب، وإنما سأمدك بأساسيات هذا المفهوم لدعم المعلومات النظرية التي ستحتاجها على امتداد أبواب الكتاب. [3]

لماذا OOP؟

بالحلبي: هيك 😒. (عد إلى مقدمة الفصل واقرأها مجددًا)

---

[1] البرمجة الإجرائية: البرمجة باستخدام التوابع فقط، دون استخدام الفئات.
[2] أكوس: أفضل، وهي اسم التفضيل من الكلمة كيّس.
[3] بإمكانك الاطلاع على كتاب "الإبحار في لغة C#.net، البرمجة كائنية التوجه OOP" لـ حسام الدين الرز للتعمق في OOP، أو كتاب "فيجوال ستوديو 2008" لـ أحمد جمال خليفة، وطبعًا كتاب "برمجة إطار عمل NET." للأستاذ الكبير تركي العسيري لا يعلى عليه إطلاقًا.

الفئات والكائنات Classes And Objects

بُنيَ مفهومُ البرمجة كائنية التوجه Objected Oriented Programming على مفهوم الكائنات، ولغات دوت نت NET. أخذت قوتها من هنا. وفي هذه الفقرة سنتناول أهم ما يجب عليك معرفته لتنشئَ أكوادًا أقوى وأكثر تطبيقًا، مع افتراض أن لديك اطلاع ولو على عناوين هذا المفهوم.

الفئات هي تعبير عام كوسائط النقل، والبشر، والعلوم، والمشاعر، وغيرها الكثير. هذه الفئات – وعلى اعتبار أنها تعابير عامة – يمكن اعتبارها صيَغًا أو أشكالًا قياسية توجد على أساسها أشياء أخرى نسميها الكائنات أو الأشياء، فالكائنات ماهي إلى نُسَخ من الفئات، أو مُشتَقّات منها. أي يمكن اعتبار المعادلة $ax^2 + bx + c = 0$ هي فئة – بطريقة أو بأخرى – لأنها تمثل شكلًا عامًّا تتواجد عليه المعادلات من الدرجة العامة، أما المعادلة $x^2 + 4x - 4 = 0$ هي كائن لأنها تمثل تخصيصًا للمعادلة، وحالة من حالاتها.

وبالمقابل فالكائنات هي شكل واقعي تمثل فئة ما وفق مجموعة من الميزات. فالتعابير العامة التي قلنا أنها فئات يمكن تخصيصها بالشكل التالي لتصبح كائنات:

- وسائط النقل هي فئات، لها خصائص مثل النوع والموديل واللون وعدد الأبواب و..، كما تتميز بمجموعة من الأفعال كالقيادة وفتح الباب والوقوف و..، كما تمر عليها مجموعة من الظروف مثل حصول حادث عليها أو دخول ركاب إليها أو وصولها لوجهتها أو غيرها الكثير والكثير، ومن الضروري جدًا أن تستطيع تمثيل الأشياء في الواقع على شكل فئة لها خصائص Properties وأفعال Methods وأحداث Events، ذلك أن هذه المفاهيم الثلاث ستعطي برامجك واقعية وتختصر عليك الكثير من الأكواد والمتغيرات وبنى الشرط. وبتخصيص هذه الفئة – أقصد وسيطة النقل – نحصل على كائنات مثل سيارة وطائرة وحافلة وسفينة وغيرها، وفي أيامنا أصبح لكل من السيارات والطائرات والسفن والحافلات وغيرها أنواع وأنواع. والفكرة التي نطرحها دائمًا: أنت لا ترى وسائط نقل، فهو مجرد مفهوم يشير إلى نوع محدد، وإنما ترى السيارات والطائرات والسفن و...، وأعتقد أن احتمالية فهمك لهذه الفكرة الآن – ومستواك هذا – أعلى مما كان عليه عندما قرأتها في كتابي السابق.

- أما البشر، فمنهم جنسيات وألوان وأعراق وأديان.
- والعلوم كذلك الأمر هي مفهوم عام لا يمكن مصادفته أو التعامل معه، أنت تتعامل مع علم الرياضيات، وعلم اللغة، وعلم الهندسة، وعلم البرمجة، وعلم الفلسفة. والمشاعر أيضًا هي مفهوم عام.

قد تسألني: أين سأستفيد من هذا الكلام في برامجي؟؟ وسأجيبك: **لا أعلم!!**

يمكن إنشاء فئة بالصيغة التالية:

```
class ClassName
{
    //أكواد الفئة
}
```

أما بالنسبة للكائنات فيمكن اشتقاقها (أو تخصيصها) بالشكل التالي:

```
ObjectName = new ClassName();
```

حيث ClassName وObjectName هي اسم الفئة واسم الكائن المشتق من هذه الفئة.

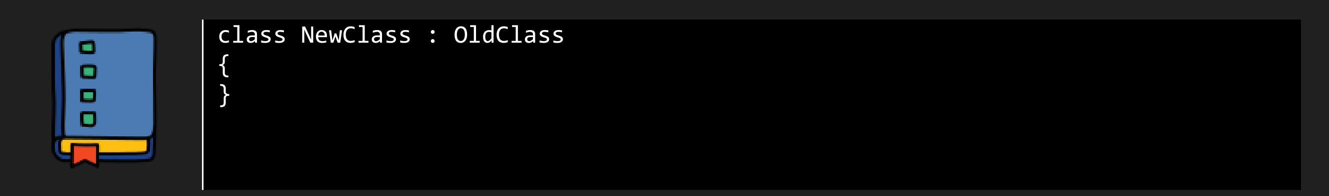
مبادئ OOP الأساسية

من المهم أن تكون على دراية بالأفكار والشعارات التي تنادي بها OOP، وينبغي عليك التوسع بهذه المفاهيم، والتي قد تضفي على برامجك سهولة وقوة في الوقت ذاته.

أهم مبدأ من مبادئ OOP – برأيي على أقل تقدير – هو الوراثة Inheritance، ومن خلالها بإمكانك اختصار وقت وجهد كبيرين أثناء إنشاء برامجك. وفيها شيء من الاشتراكية [1] ولو

[1] الاشتراكية: مصطلح سياسي يشير إلى التشاركية في الموارد والأملاك، وبناءً عليه فإن الجهات الحاكمة لها الحق بمصادرة أملاك الأفراد ومواردهم "إذا ما كان للمجتمع فائدة بذلك".

بقيود، وفيها يتم نسخ جميع محتويات فئة ما إلى فئة أخرى، لتتمكن من إنجاز تطبيقات كبيرة بخطوات وعمليات أقل. يمكن توريث فئة ما لفئة أخرى من خلال الصيغة:



```
class NewClass : OldClass
{
}
```

حيث NewClass هي الفئة الجديدة التي سترث فئة أخرى، وOldClass هي الفئة القديمة التي سيتم الاشتقاق منها.

تتميز OOP أيضًا بمفهوم التغليف Encapsulation، والذي يعتمد على تجميع عدة أكواد – قد تكون عشرات أو مئات – وتغليفها تحت بند معين لتحصل على نتيجة – أو نتائج – محددة. تمامًا كالأجهزة والآلات مثلًا، لا يهمك كيف تم تجميع القطع الصغيرة والبراغي والأجزاء مع بعضها لتشكيل محضّرة قهوة، لكن يهمك الحصول على القهوة بشكل أو بآخر. أي أن الطريق المسلوك غير مهم، وإنما النتيجة.

وعادةً ما يُستفاد من مفهوم التغليف عند العمل كمجموعات، حيث يتخصص كل مبرمج بمجال محدد، أو عند توزيع الأكواد من خلال ملفات DLL أو خدمات API.

ومن المفيد في هذا السياق أن أقتبس التالي: "باختصار شديد لنفترض نظام محاسبي يتضمن نظام لإدارة المخازن، وآخر للصيانة، وثالث من أجل المبيعات.

في هذه الحالة يدعوك مبدأ الـ Encapsulation ليكون لكل واحد من هذه النظم عدد محدد من الدوال للدخول والخروج من هذا النظام والتي يمكن للنظم الثلاثة التواصل من خلالها، فمثلًا في نظام المبيعات تجد (إضافة عملية مبيعات) (إضافة مشتريات) (خصم) ... إلخ، في مجموعات محددة جدًا من الدوال.

طبعًا لو لاحظت أن العملية مثل إضافة عملية مبيعات تتطلب طابورًا من الأوامر، يتضمن فتح قاعدة البيانات والتأكد من أن البيانات المدخلة صحيحة والتأكد من وجود الكمية ومن ثم تخزين الناتج في قاعدة البيانات، ثم تجميعها في النهاية على شكل أمر واحد يقوم

زميلك الآخر الذي يقوم ببرمجة واجهات المستخدم إلى استخدامه بدلًا من الغوص في كل هذه التفاصيل الفنية.

كما لاحظت، يفيد هذا الموضوع الأشخاص التي تعمل في مجموعات أولًا، حيث لن أكون مضطرًا لفهم كودك بالكامل ويكفيني أن أعرف كيف أتعامل معك، كما أن الكود سيكون مصممًا على شكل هرم حيث كل أمر يستتبعه مجموعة من الأوامر، لكن سيكون زميلك المبرمج قادرًا فقط للوصول إلى رأس الهرم وهو ما يقلل كثيرًا من الأخطاء، كما يحمي متغيراتك الخاصة من العبث بها عن طريق الخطأ من المبرمجين الآخرين." كما قال أحمد جمال خليفة في كتابه [1].

أما الأستاذ تركي العسيري فقد قال: "يقصد بالتغليف Encapsulation في لغات OOP بوضع جميع الاشياء معا Putting everything together، بحيث تحقق استقلالية الكائن المطلقة ببياناته الخاصة به وحتى أكواده، من المزايا التي يقدمها لك التغليف هو امكانية تطوير البنية التحتية للكائن بدون ان يتأثر تركيب برنامجك ودون الحاجة الى تعديل سطر واحد من أكواد البرنامج، مثلا لو قمت بتصميم فئة للبحث عن الملفات واعتمدت عليه بدرجة كبيرة في برنامجك، وبعد فترة من الاختبارات والتجارب القوية لاحظت بطء في عملية التنفيذ، فكل ما ستفعله هو تعديل البنية التحتية للفئة الخاصة بالبحث وتطوير خوارزميات أكوادها دون تغيير سطر واحد من سطور البرنامج الاخرى والتي تستعمل هذه الفئة بالتحديد.

كلما زادت استقلالية الفئة، كلما زادت كفاءة اعادة استخدامها في برنامج آخر وتطبيق اسلوب اعادة استخدام الأكواد Code Reusability. مبدأ اعادة استخدام الأكواد من أحد المبادئ الضرورية التي يتوجب عليك محاولة التعود على تطبيقها دائما في برامجك ومشاريعك اليومية، بحيث تتمكن من الاستفادة من الفئة التي صممتها في أكثر من مشروع وأكثر من برنامج. وحتى تنشئ فئة قابلة لإعادة الاستخدام، حاول دائما وقبل ان تبدأ بكتابة سطر واحد من الفئة بأخذ احتياطاتك للمستقبل واسأل نفسك أسئلة شبيهه بـ: كيف يمكنني الاستفادة من هذه الفئة في برنامج آخر؟ كيف اسمي واحدد

[1] انظر كتاب "خطوة بخطوة مع فيجوال ستوديو 2008" لـ أحمد جمال خليفة (ص 160).

الخصائص، الطرق والاحداث بحيث تكون قابلة للعمل مع أكثر من برنامج وقابلة للتطوير ايضا؟ كيف اجعل هذه الفئة مستقلة قدر المستطاع عن أي أكواد او كائنات اخرى في البرنامج بحيث يمكنني اســتخدامها في برنامج آخر؟... الخ من الاســئلة والاعتبارات التي لابد من وضعها في الاعتبار قبل بناء الفئة وعند كتابة كل اجراء من اجراءاتها." [1]

الأمر الثالث الذي تقدمه لك OOP هو تعدد الواجهات أو تعدد الأشـكال Polymorphism، والذي شرحته في كتابي الأول بأسلوب فاشل جدًّا 🤦🏻‍♂️، لدرجة أنني كلّما قرأت ذلك الشرح رجّحت أنني وقتها كنت أشـم شعلة 👽 [2]. وهنا سأشرح الفكرة بأسلوب أفضل (ولو أن هذا الكتاب وبالتحديد هذا الفصل ليس لشرح الأفكار النظرية بقدر ما هو لعرضها):

يمكنك كبداية إنشاء فئة مجرّدة – لا يمكن استنساخ كائنات منها – ووضع مجموعة من الطرق والأحداث والخصائص فيها، ثم ننشئ فئات أخرى نورثها هذه الفئة المجردة. والغاية من هذا الأسلوب هو إنشاء مفاهيم برمجية مشتركة بين فئات مختلفة. وبكلام أوضح: لدى أغلب أدوات ويندوز القياسية – مثل TextBox وButton وLabel وغيرها – خاصيات مشتركة مثل Text وLeft وTop وName وغيرها، بالإضافة إلى أحداث عديدة مشتركة مثل Enter وMouseDown والكثير أيضًا، مع أن هذه الأدوات من طبيعة مختلفة، فلكل أداة نوع محدد يتمثل بفئة محددة لها معنى محدد وغاية محددة لوجودها. فهذه الأدوات – والتي تعتبر فئات بطبيعة الحال – مشتقة من فئة مجردة هي فئة Control.

وقد قال الأستاذ الكبير تركي العسيري في كتابه برمجة إطار عمل فيجوال بيسك دوت نت: إن الواجهة Interface [3] هي مجموعة من الطرق والخصائص والأحداث التي تصف فئة معينة، ومن مبادئ OOP مبدأ تعدد الواجهات Polymorphism [يُخيَّل إليك بالبداية أنه الاسم الكيميائي لدواء السعلة 👨🏻‍🔬😂] وهو احتواء الفئة الواحدة أكثر من واجهة تصفها.

---

[1] انظر كتاب "فيجوال بيسك للجميع" لـ تركي العسيري (ص 151).
[2] شم الشعلة هو ظاهرة انتشرت في سوريا في ظلّ الحرب التي شردت أهلها ورفعت الأسعار بشكل جنوني، والشعلة هي مادة كيميائية سامة هي نفسها الغراء (مادة لاصقة)، والتي أدمنها الأطفال بشكل خاص – المشردين منهم وغير المشردين – لا سيّما أن سعرها ليس مرتفعًا كالمُدمّنَات الأخرى كالمخدرات وغيرها.
[3] الفصل الخامس فيه نبذة عن الواجهات، كما أن الفصلين السادس والسابع يحويان تطبيقات على الواجهات.

وكمثال على تعدد الواجهات افترض لو أنه لديك مجموعة من ملفات النصوص المختلفة كـ txtFiles وrtfFiles وhtmlFiles وxmlFiles، وبالتأكيد فإن وظائف هذه الملفات تختلف عن بعضها اختلافًا كبيرًا، ولكن الواجهات التي تحتويها متشابهة ومتشاركة، فيمكن مثلًا تعريف خاصية تمثل اسم الملف FileName أو حجمه FileSize، وطريقة تقوم بعملية الحفظ ()Save وأخرى بالحذف ()Delete، حيث يمكن تطبيق جميع هذه الواجهات على الفئات المختلفة.

فلو قمت بتطوير إجراء يقوم بحفظ الملف النصي فستنشئ وسيطًا يستقبل كائنًا من النوع txtFile:

```
void Save_txtFile(txtFile F)
{
    …
    F.Save();
    …
}
```

ونفس الشيء ستفعله مع الملفات الأخرى:

```
void Save_rtfFile(rtfFile F)
{
    …
    F.Save();
    …
}
```

```
void Save_htmlFile(htmlFile F)
{
    …
    F.Save();
    …
}
```

```
void Save_xmlFile(xmlFile F)
{
    …
    F.Save();
    …
}
```

لا بل الأكثر من ذلك، يتوجب عليك عند استدعاء إجراء الحفظ كتابةُ الجمل الشرطية في كل مرة تود فيها التحقق من نوع الملف، وعلى ضوء المقارنة نقوم باستدعاء إجراء الحفظ المناسب لنوع الملفات المطلوب، كما أنه عليك الإعلان أيضًا عن جميع الكائنات التي تمثل الفئات المختلفة لإرسالها إلى الإجراء المختص:

```
txtFile TF = new txtFile();
rtfFile RF = new rtfFile();
htmlFile HF = new htmlFile();
xmlFile XF = new xmlFile();
…
…
switch(FileType)
{
    case FileType.txtFile:
    {
        Save_xmlFile(TF);
        break;
    }
    case FileType.rtfFile:
    {
        Save_xmlFile(RF);
        break;
    }
…
}
…
```

والأنكى من ذلك [1] أنه لو أردت إدراج نوع بيانات جديد إلى برنامجك فعليك إضافة ذلك إلى بنى الشرط وقسم التصريحات وقسم الإجراءات (كإجراء الحفظ)، أي أنك ستعيد كل ما فعلته سابقًا بالنسبة لأنواع الملفات الأخرى!!

أما مع تعدد الواجهات فالموضوع أبسط بكثير، فكل ما ستفعله هو تعريف واجهة باسم IFile مثلًا، وتعريف إجراء يقوم باستقبال كل أنواع الملفات التي تحتوي على الواجهة IFile:

```
void SaveFile(IFile F)
{
    …
    F.Save();
    …
}
```

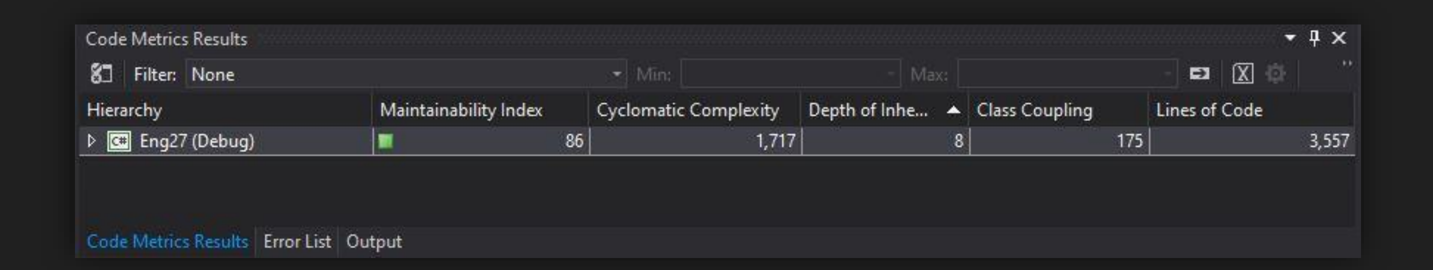
[1] الأنكى من ذلك: الأبشع (أو الأسوأ) من ذلك، أو: الأكثر من ذلك.

لا داعي الآن لبنى الشرط، في حين أنك تحتاج لتعريف الكائنات الممثلة عن الملفات، تخيل مقدار الاختصار والتبسيط الذي قمت به في الكود!! للمزيد راجع كتاب برمجة إطار عمل.[1]

الكود النظيف في البرنامج النظيف

الكود النظيف Clean Code هو الكود سهل القراءة والفهم والتعديل. الكود النظيف هو الكود البسيط، هو الكود الذي يمكنك الوصول إليه بأكسل الطرق، وأصحها. فدرجة تعقيد الكود تزيده سوءًا ولا تجعل منك محترفًا 👨🏻‍💻!

يعطيك الفيجوال ستوديو أداة لقياس قوة الكود Code Metrices، بقياس بساطته وقابلية صيانته ومدى ارتباط فئاته ببعضها:



ما هي الأمور التي تجعل الكود نظيفًا؟ الفقرات التالية ستجيبك على هذا السؤال.

سمّ متغيرات برامجك بأسماء ذات معنى

إذا لم تكن معتادًا على المشاريع الكبيرة، فإنك قد لا تجد مشكلة في تسمية متغيرات برامجك، أول اسم يخطر في بالك (وأحيانًا حروف) ستختاره لتسمية المتغيرات، خصوصًا إذا لم تكن بحاجة لفتح المشروع إلا مرات معدودة.

[1] انظر كتاب "برمجة إطار عمل فيجوال بيسك دوت نت" لـ تركي العسيري (ص 187). وقد نقلت بعضًا من محتوى الفقرة التي تتحدث عن الواجهات حرفيًّا تقريبًا.

أما إذا كنت قد جربت المشاريع الكبيرة، أو العمل ضمن فريق، فإن أسماء المتغيرات ستكون معضلة بحد ذاتها، خصوصًا إذا كانت المتغيرات لا تصفها كلمة أو اثنتين وإنما جملة أو أكثر!

وعليه، سواء أكنت معتادًا على المشاريع الكبيرة أو لا، تعود على استخدام أسماء متغيرات تشير لوظيفة المتغيرات، واستخدام الاختصارات. ومن العادات الجيدة أن تستخدم أسماء المتغيرات بأحرف صغيرة (تفصل بين الكلمات بشارطة _)، وأسماء الخصائص بأحرف كبيرة (لا داعي لفصل الكلمات بشيء).



```
string n; // name of user
```

لا يعتبر هذا الكود نظيفًا، فلو لم يوجد التعليق لما كانت الغاية من المتغير معروفة، وهذا لن يحدث فقط مع من سيقرأ كودك، وإنما ستحدث معك أيضًا مع الأيام، بعد أسابيع أو أشهر أو حتى سنوات عندما تعود للكود وتقرأه وتتحزّر ماذا كانت وظيفته!

يمكن جعل الكود أفضل وأوضح بتسميته name_of_user، أو name ببساطة، أو username أو غيرها من الأسماء التي تدل على وظيفة المتغير (لا تستخدم دائمًا أسماء متغيرات مختصرة أو مقتضبة، فقد تتسائل مثلًا هل المتغير name يشير إلى اسم المستخدم أم اسم الزبون مثلًا).

طبق مبدأ المسؤولية الواحدة SRP

تعتبر الفئات والطرق أساليب جيدة لتنظيم أكوادك في أي لغة برمجة، إلا أن كثيرًا من المبرمجين يهملونها أو يكتبونها بطرق سيئة، خصوصًا المبتدئين منهم (قد تجد بعض المتمرسين أيضًا لا يهتمون بتنظيف أكوادهم والاعتناء بفئات برامجهم وطرقها)، فتجد أحدهم يكتب تابعًا يقوم بكل شيء في البرنامج، مما يجعل الكود صعبًا للقراءة، والفهم، والصيانة! فحتى لو كنت محترفًا لوجدت صعوبة في قراءة أسطر هذا الكود، ولحصلت على مشاكل عند محاولة إصلاح أخطاءه.

لتحقق مبدأ المسؤولية الواحدة عند إنشاء طريقة ما، ضع في ذهنك ما يلي:

- يجب أن تكون الطريقة صغيرة.
- يجب أن تقوم هذه الطريقة بشيء واحد، ويجب أن تقوم بهذا الشيء على أكمل وجه.

وكما هو الحال مع المتغيرات، سمّ فئات وطرق برامجك بأسماء مفهومة وذات معنى، وابدأها بأحرف كبيرة، حتى إذا كان الاسم يحوي أكثر من كلمة فإنك لن تحتاج للفصل بين الكلمات، إذ إنها تبدأ بأحرف كبيرة. كما أن استخدامك للتوثيق سيزود أشياءك البرمجية بدليل استخدام مصغّر. الفصل الثاني سيفصّل التوثيق Documentation بشكل جيد.

```
int Add(int X, int Y)
{
    return X + Y;
}
```

قد تتسائل عن الفائدة وراء إنشاء تابع يقوم بوظيفة سطر برمجي واحد، بحجة أننا لم نختصر شيئًا من الأكواد في أجزاء البرنامج، وإنما استبدلنا عملية الجمع بتابع يقوم بذلك. ولكن في الحقيقة الفائدة التي حققناها كبيرة، فإذا كان البرنامج يحوي هذه العملية في أكثر من موضع، ربما عشرات المواضع أو مئات المواضع، وأردت تعديل هذه العملية، فعليك تعديلها في كل المواضع التي تتواجد فيها هذه العملية. أما لو أنشأت طريقة تمثل العملية، فإنك ستذكر العملية في المواضع التي تحتاجها فيها، وعند القيام بتعديلها فإنك ستعدلها مرة واحدة فقط. كنت قد نشرت سابقًا مقالًا على منصة الباحثون المسلمون عن التوابع، يمكنك الرجوع إليه للمزيد حول التوابع، من هنا [1].

لا تكرر أكواد برنامجك، إذا كان لديك سطر برمجي أو أكثر تستخدمه في أكثر من مكان من برنامجك (أو تتوقع ذلك)؛ ضعه في طريقة، إذا كان لديك أكثر من طريقة متعلقة

[1] راجع منصة الباحثون المسلمون
https://muslims-res.com/%d9%85%d8%a7%d8%b0%d8%a7-%d9%8a%d8%b9%d9%86%d9%8a-%d8%a7%d9%84%d8%aa%d8%a7%d8%a8%d8%b9-%d9%81%d9%8a-%d8%a7%d9%84%d8%a8%d8%b1%d9%85%d8%ac%d8%a9%d8%9f/

ببعضها؛ ضعها في فئة، إذا كان لديك أكثر من فئة متعلقة ببعضها؛ ضعها في مجال أسماء.

التعليقات في الكود مثل الملح في الطعام، كثيره مثل غيابه

لا تكثر من التعليقات، فكثيرها مربك، وخير الكلام ما قل ودل، كما أن كثيرًا مما ستكتبه ضمن التعليقات يمكن تحصيله بتسمية الفئات والطرق والمتغيرات بشكل جيد وذي معنى. ولكن بالوقت نفسه، لا تحرم برامجك من التعليقات.

طبق مبادئ SOLID

مبدأ المسؤولية الواحدة SRP هو أحد مبادئ SOLID (في الواقع هو أولها)، وتدور كلها حول فكرة الكود النظيف Clean Code.

لا يسعنا شرح هذه المبادئ في هذا الكتاب، وإنما أحببت التنويه لها حتى تُراجَع. وخير كتاب عربي يمكنك الرجوع إليه لفهم هذه المبادئ كتاب للأستاذ خالد السعداني "المبادئ السوية لإنشاء برامج قوية"، وهو كتاب خفيف جميل ماتع ممتع، من أفضل الكتب التي قرأتها يومًا، قد تجده أفضل من الكتب الأجنبية حتى!

الفصل الأول – ما هي الـ UI وUX؟

تهتم UI (واجهة المستخدم User Interface) بشكل ومظهر البرامج وبالتصاميم، وتعتمد بشكل كبير على الصور والأشكال الرسومية، ويُجذَب المستخدم – المبتدئ – بها غالبًا، فهي حرفيًّا كالمكياج بالنسبة للإناث 🐍💁🏼.

أما UX (تجربة المستخدم User Experience) فتعتمد على الإحصائيات والتقارير والخبرات والدراسات، وما يسمى بـ Feedback (حرفيًّا هي التغذية العكسية، أو العائدة، وتسمى أيضًا الرابطة العائدة العكسية، ومن خلالها يمكنك معرفة ما يجب أن تعدّل على المُدخلات لتحصل من خلاله على نتائج معينة، وبشكل مشابه يستخدم هذا المصطلح في البرامج والمواقع لمعرفة رأي المستخدم والمشاكل التي واجهته، لتصحيحها، للحصول على النتائج المطلوبة)، كما أنها تتعلق بالهندسة الاجتماعية.

مفهوم الـ UX أوسع من الـ UI، إذ إنه يحتاج إلى دراسات وأبحاث وخطط وخوارزميات تقود برنامجك للحصول على المطلوب. أما UI فهي تقتصر على التصاميم واختيار الألوان والخطوط والأشكال والصور المناسبة، وهناك منصات عمل Frameworks جاهزة تعطيك واجهات مستخدم عصرية وحديثة، والتي سنناقشها في الفصول الأخيرة من الكتاب.

سر نجاح تصميم التطبيقات والمواقع ليس الفكرة الجيدة والمبتكرة فقط (وقد تكون)، وإنما جودة تصاميم واجهات المستخدم UI وتجارب المستخدم UX. لا يهم مدى جودة فكرتك إذا كان تصميمك سيئًا أو صعبَ الاستخدام، خصوصًا للمستخدمين ذوي الخلق الضيق[1]. [2]

بشكل عام يمكنك القول أن: [3]

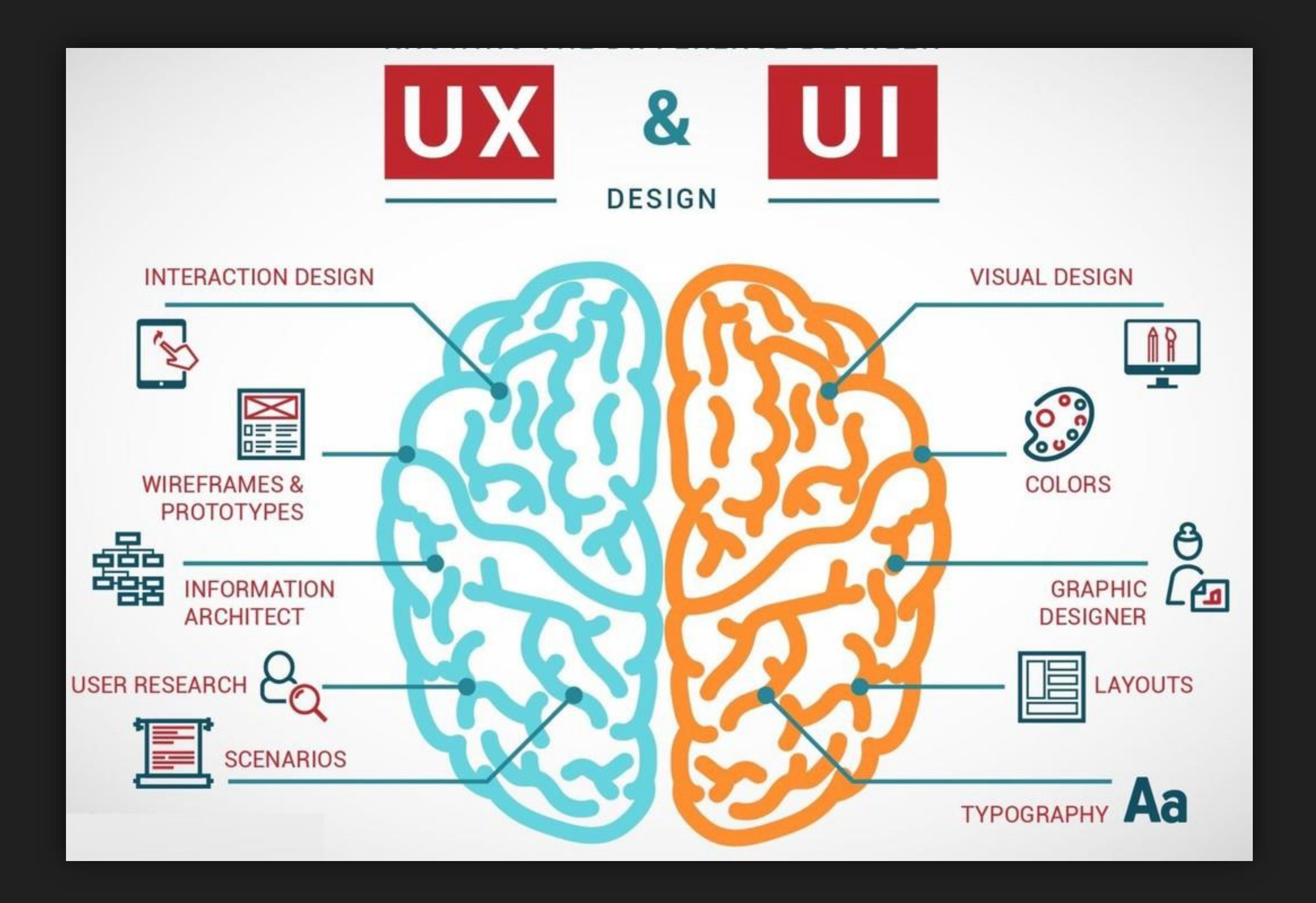
- مصمم UX أشبه بالمعماري (أو بمعنى أدق: بالمهندس المدني)، فهو ينشئ البنية التحتية للمنتج ويساعد في تطوير الأمور المحسوسة فيه، يهتم بالمستخدم واحتياجاته، يساعده في الاستفادة من المنتج بأكبر قدر ممكن، يفهم ويحلل سلوك المستخدم ونفسيته.
- مصمم UI أشبه بمهندس الديكور، فهو يهتم بشكل المنتج وواجهاته، وكيف تنعكس هذه الواجهات على العلامة التجارية بك كصاحب المنتج، يساعد في تطوير الأمور غير المحسوسة في المنتج (مثل هل البرنامج مريح، جذاب، لطيف، أنيق، ...). مصمم UI عنده ذوق في الألوان وملائمة الألوان المختلفة مع بعضها.

من الممكن أن يكون مصمم UI وUX هو شخص واحد يقوم بعملية التصميم بأكملها.

---

[1] ذوو الخلق الضيق: قليلو الصبر.
[2] راجع موقع Medium
https://medium.com/swlh/ui-ux-design-guide-with-terms-explanations-tips-and-trends-754b9356d914
ضمن فقرة "مصادر إضافية" في هذا الرابط هناك روابط عديدة مفيدة تعطيك تقنيات جيدة للحصول على تطبيقات قوية، أحيلُك إليها لمزيد من التفاصيل.
[3] المصدر السابق نفسه.

الصورة التالية تختصر لك أهم الفوارق بين UI وUX:



كما هو واضح فالـ UI يهتم بالمرئيات، كالتصميم المرئي من حيث الأدوات المستخدمة ووجود الصور والأيقونات وتناسقها مع شكل برنامجك (بغض النظر هل هذه الصور والأيقونات مفيدة أم لا، فالـ UI يهتم بالشكل لا أكثر)، ويهتم أيضًا بألوان الأيقونات والخطوط، وبتصميم الرسومات كأشكال الأدوات المستخدمة مثلًا (لاحظ شكل أدوات برنامج وورد وقارنها مع الرسام مثلًا، أو قارن بين النسخ المختلفة من نفس البرنامج مثل الوورد مثلًا)، وبمظهر تطبيقك عمومًا (لاحظ تطبيق فيسبوك، وتويتر، وبريد غوغل، وواتساب وغيرها، لكل تطبيق مظهر عام يميزه عن التطبيقات الأخرى، وقد تتشابه التطبيقات مع بعضها مثل تلغرام وواتساب ومع ذلك هناك من يفضل أحدهما عن الآخر، وهكذا..)، وبالخطوط وأشكالها.

أما UX فيهتم بالهدف والنتيجة أو فلنقل بالمضمون، كالتصميم التفاعلي، والنماذج الأولية للتطبيق، وشبكة المعلومات فيه، ودورة حياة البرنامج!

وإن صحّت الصورة فالجزء الأيمن من الدماغ مسؤول عن الأمور الفنية لذلك فهو من يقيّم الـ UI في التطبيقات، والأيسر مسؤول عن الحسابات والتفكير لذلك فهو من يقيّم الـ UX.

الصورة التالية تقارن بين UI وUX بطريقة أخرى:

naqrah
واجهة الاستخدام UI
- التناسق
- التصميم البصري
- انشاء الهوية

تجربة الإستخدام UX
- الهيكلية
  تشكيل الواجهة
  تصميم التنقلات
  تصميم المحتوى
- البنية
  التصميم التفاعلي
  هيكلية المعلومات
- المجال
  برمجيات النطاق
  متطلبات المحتوى
- الاستراتيجية
  حاجات المستخدم
  مهمة الموقع / التطبيق

www.naqrah.net

وواضح منها أن UI يهتم بالشكليات، في حين UX يهتم بالمضامين كما بيّنا سابقًا.

بالإضافة لما سبق، فإن مصمم UX – وعلى سبيل المثال – يهتم بمكان وضع الزر button بحيث يجده المستخدم بسهولة، في حين أن مصمم UI يفكر بكيفية جعل الزر جميلًا وأنيقًا بحيث يجعل المستخدم يرغب بالنقر عليه!!

وختامًا، نصيحتي – وأمنيتي وطلبي منك – هي التحلي بذوق فني راقٍ، غير مزعج للعين، غير مرهق للدماغ، بحيث تعطي المستخدم راحة عند التنقل بين واجهات وأقسام برنامجك، وتساعده في معرفة ما له وما عليه في البرنامج باستخدام أقل الأساليب الممكنة.

الألوان

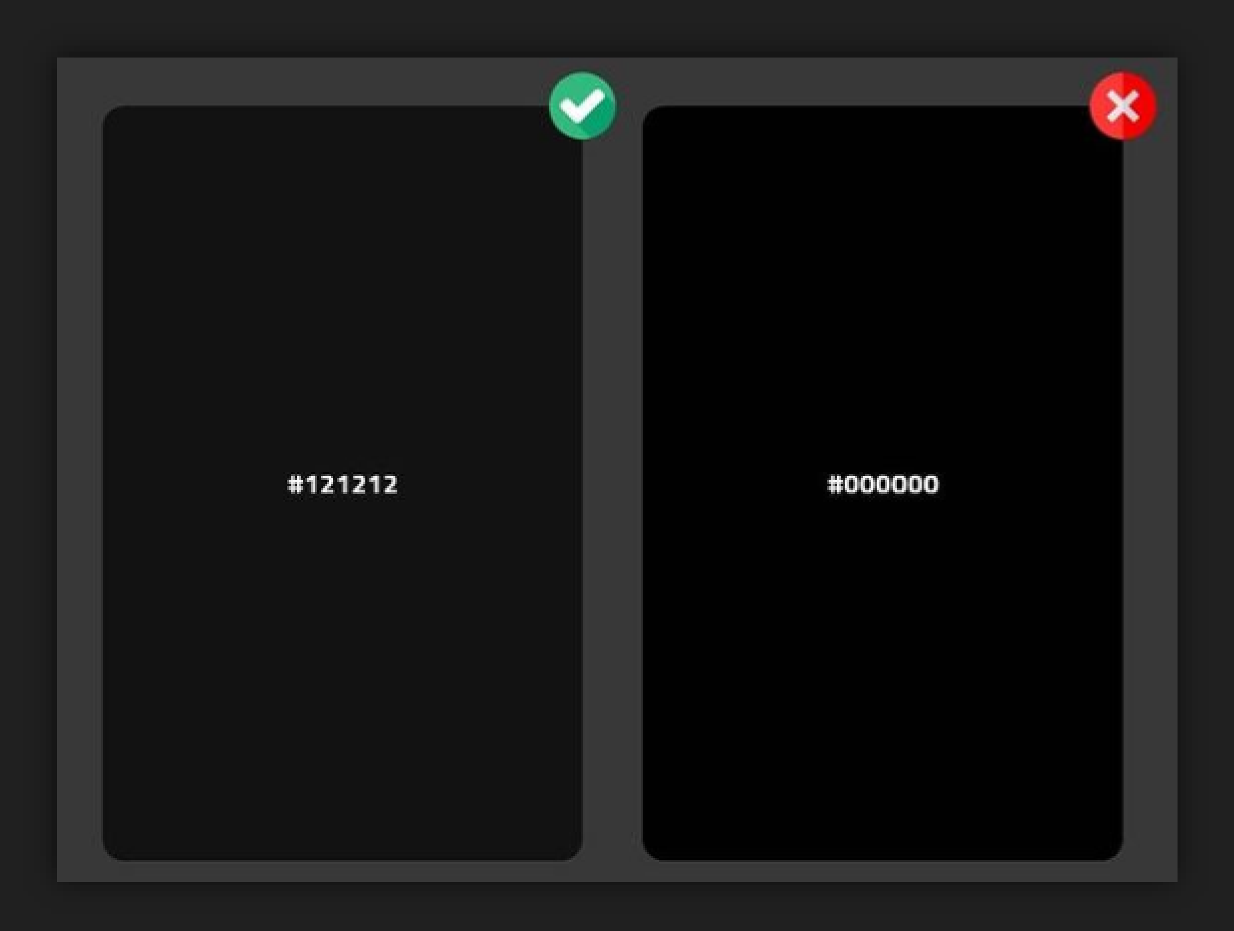
الغاية من الألوان تمييز المحتوى عن بعضه، لذلك فبضبطك للألوان يمكنك التلاعب بعقل المستخدم، وعليه ستجذبه لتطبيقك/موقعك أو تنفره منه.

لا تكثر من الألوان في برنامجك (لا تجعل نوافذ برنامجك أشبه بفتاة جديدة على المكياج رجاءً)، استخدم ألوانًا محددة متباينة متعلقة ببعضها، ذات غاية وهدف من وجوده.

بالنسبة للخطوط اضبط العناوين العريضة بلون ما، العناوين الفرعية بلون آخر (لكن بوزن [1] أقل)، خذ الكتب على سبيل المثال، يمكنك تمييز العناوين عن المحتوى العادي من الألوان وأوزان الخطوط دون الحاجة لكتابة "هذا عنوان عريض" و"هذا عنوان فرعي"، و"هذا محتوى عادي".

إذا أردت جعل تطبيقك ليليًّا Dark Mode، لا تستخدم اللون الأسود، وإنما الرمادي الغامق، تمامًا كصفحات هذا الكتاب:

[1] وزن الخط هو سماكته.



يمكنك الاعتماد على بعض القواعد لاختيار ألوان نوافذ تطبيقاتك (بما فيها من أدوات وخطوط)، والتي أنقلها من صفحة Zeroone Pro من الفيسبوك، والتي تنص على ما يلي:

- قاعدة المقابل Complementary:

اللون المقابل للون الرئيسي الذي اخترته هو لون مناسب لاستخدامه معه.



- قاعدة التناظر Analogous:

اللونين المجاورين للون الرئيسي الذي اخترته مناسبين لاستخدامهما معه.



بعد اختيار اللون الرئيسي يمكنك اختيار الألوان الفرعية من خلال قاعدة المثلث أو المستطيل (أو غيرها):





النسق Layout

ذكرنا سابقًا أن أغلب التطبيقات والمواقع تنضوي تحت نسق Layout معين، وهذا مما يريح المستخدم لاعتياده على نسق معين، وإن تغيرت عليه بعض الحيثيات والتفاصيل إلا أن الشكل العام نفسه. يمكنك إنشاء نسق جديد أو عدم الاعتماد على نسق أصلًا، لكن هذا سيجعل استخدام تطبيقك أصعب.

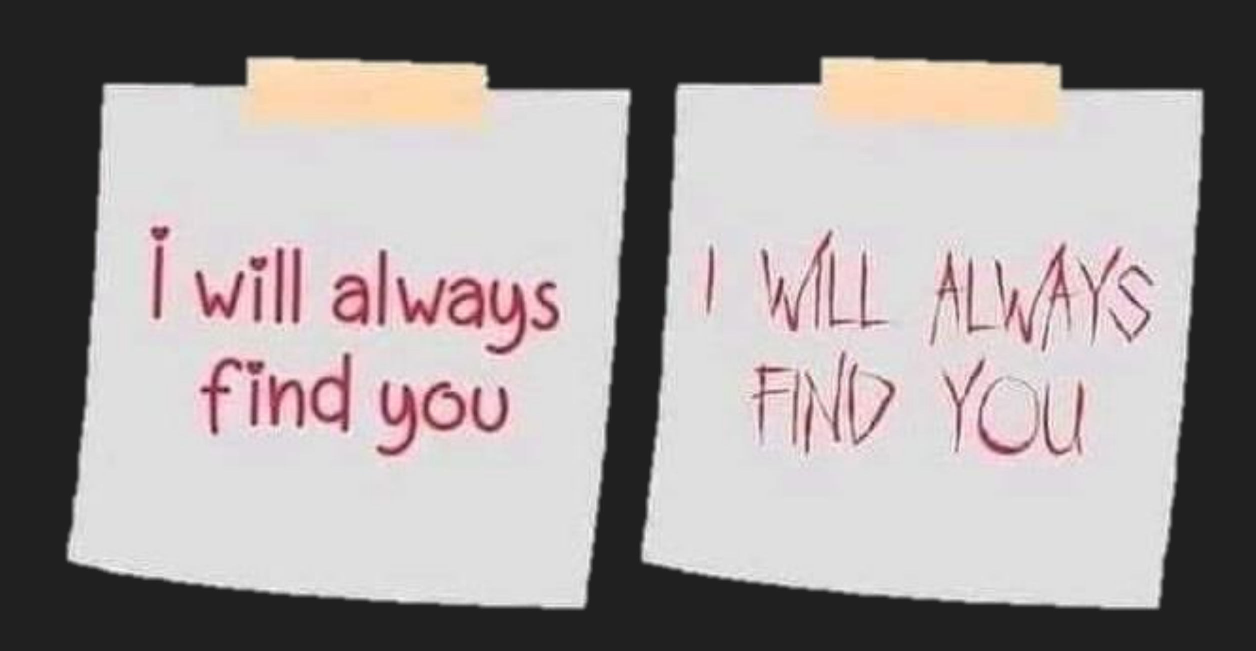
لا تقترب من حواف النافذة، أبعد أدواتك عن الحواف بمقدار ما، اجعل جميع أدواتك تبعد عن الحواف المقدار ذاته (هامش ثابت لكل الأدوات)، لا تترك مسافات خالية كبيرة في النافذة (إلا إذا كان مظهر النافذة والغرض منها يقتضي ذلك)، ولا تنس أن فيجوال ستوديو

يساعدك في عملية تصميم الواجهة كثيرًا، عند تحريك أي أداة يعطيك هوامش تلقائية بين الأداة المحركة والأدوات الأخرى وحواف الحاويات، كما يعطيك خطوطًا تضبط الأداة مع الأدوات الأخرى، استغل هذه الفرصة بشكل جيد، لا تجعل مواقع أدواتك عشوائيًّا (إلا إذا كانت العشوائية لها لمسة فنية).

الخطوط

استخدم أقل عدد ممكن من الخطوط (كأشكال وأوزان)، حاول الموازنة بين الخطوط ووظائفها (عناوين رئيسية بوزن كبير مثل اسم البرنامج أو النافذة، نصوص عادية بوزن عادي مثل العبارات المكتوبة هنا وهناك في النافذة والتي تمثل عبارات تخبر المستخدم عن وظيفة الأدوات أو الغاية منها، أو عبارات تنبيهية أو تحمل وظائف ثانوية مثل تلك التي تسأل المستخدم هل نسي كلمة السر أو التي تخبره بالنوافذ أو الصفحات ذات الصلة بالنافذة أو الصفحة الحالية، ...).

استخدم خطوطًا مناسبة، تأمل:



النصوص [1]

اجعل تطبيقك متناسقًا بمضمونه، لا تكثر من المصطلحات الدالة على المضمون ذاته، إذا استخدمت عبارة "Buy now" في مكانًا ما من تطبيقك، لا تستخدم "Purchase now" في مكان آخر. فعدم التناسق (بالشكل أو بالمضمون) = التشويش على المستخدم.

اكتب بصيغة الزمن الحاضر، فبدل "سيتم تحميل الملف"، استخدم "الملف قيد التحميل".

لا تستخدم عبارات طويلة، فالمستخدم لا يقرأ محتوى النافذة أو الرسالة بالكامل – إلا عند تعسر عمل البرنامج – وإنما يمسح العبارة بحثًا عن كلمات دلالية يفهم منها الغاية من النافذة أو الرسالة ككل. لذلك فجزّء العبارات إلى أقسام إذا أمكن ذلك.

لا تستخدم المصطلحات البرمجية (مع أني شخصيًا أفضل ذلك في بعض الحالات)، فبدلًا من "خطأ بالنظام (رمز الخطأ 111)" استخدم "خطأ في الاتصال بقاعدة البيانات: كلمة السر غير صحيحة".

استخدم الأرقام بدلًا من الكلمات، فبدلًا من "لديك ثلاث رسائل جديدة" استخدم "لديك 3 رسائل جديدة" (خصوصًا إذا كان برنامجك موجها لنحويين).

اكتب باختصار وإيجاز، فبدل "يرجى التفضل بتسجيل الدخول قبل أن تتمكن من كتابة تعليق" استخدم "سجل دخولك لكتابة تعليق" (اللطف الزائد لا طعمة له).

اختصر خطوات الوصول لوظائف تطبيقك

المستخدم – المحترف خصوصًا – يحب استخدام الاختصارات وتخصيصها إن أمكن. من الجميل في بدايات المستخدم مع تطبيقك تجربة قوائم ونوافذ التطبيق للوصول إلى وظيفة ما منه إذا كان تصميمه متقنًا ومناسبًا، لكن الموضوع يصبح لاحقًا روتينيًّا مملًا مع تكرار استخدام البرنامج، لذلك عليك تخصيص الاختصارات لقوائم ووظائف برنامجك المختلفة، وإضافة أشرطة هنا وهناك تضع فيها أكثر وظائف التطبيق أهمية فيه، وسيكون

[1] منقول بتصرف من صفحة بلورة ديزاين على فيسبوك.

من اللطيف منك تمكين المستخدم من تخصيص هذا الشريط. كما يفضل أن تستخدم مفاتيح اختصار مشهورة ومألوفة للمستخدمين.

القوائم الطويلة مملةٌ

جزّء فقرات برامجك لأقسام، القوائم الطويلة تربك المستخدم وتشتت انتباهه، لاحظ الأمثلة:

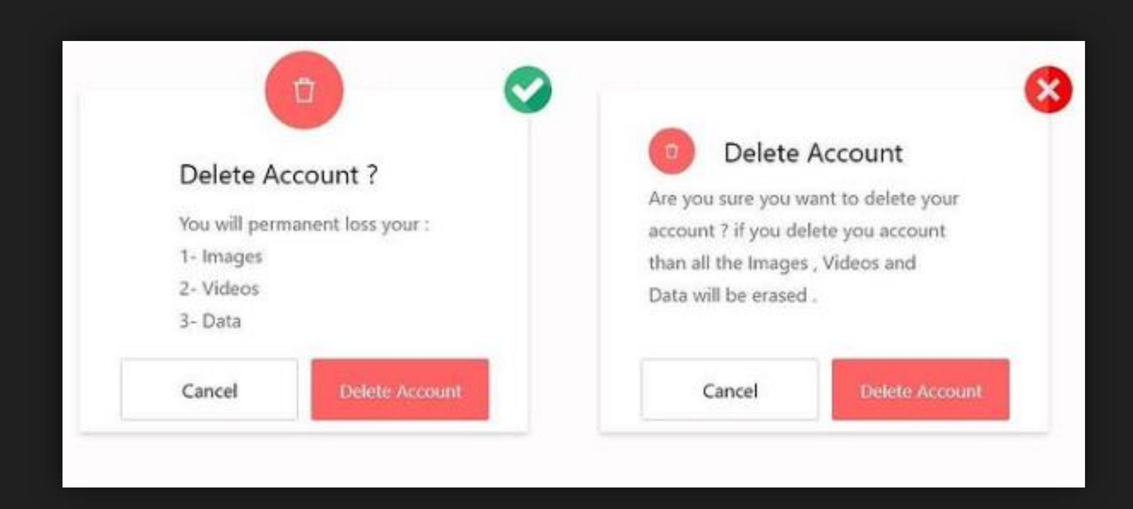
مثال١: تطبيق تلغرام يعطيك جميع المحادثات بقائمة واحد بينما تلغرام بلس يفصل بين المستخدمين والمجموعات والقنوات وغيرها، مما يسهل تصفحها.

مثال٢: البحث ضمن قائمة المنشورات المحفوظة في فيسبوك مجهد وممل (خصوصًا أنها تستغرق بعض الوقت ليتم تحميلها)، بينما البحث في منشورات قائمة معينة أسهل.

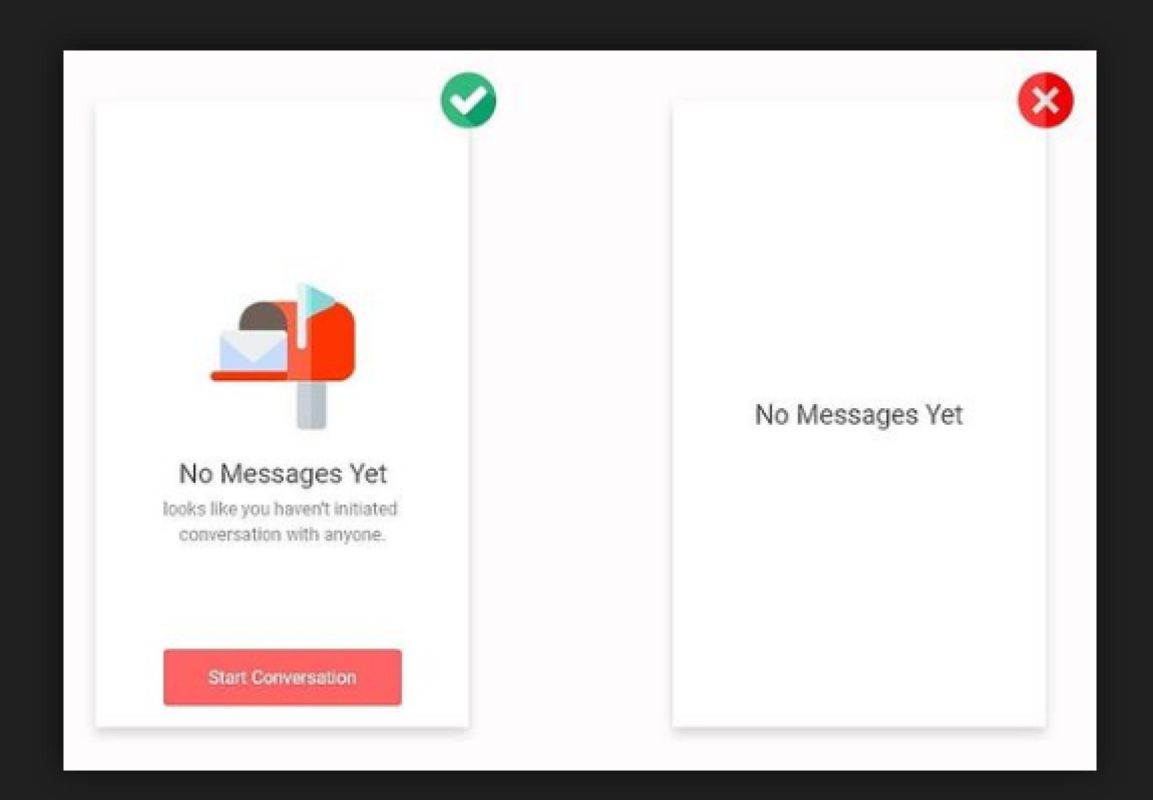
يظهر هذا الشيء جليا عند البيانات الكبيرة، التي يمكن أن تصل للعشرات او ان تتجاوز المئة!

اعتن بالتفاصيل

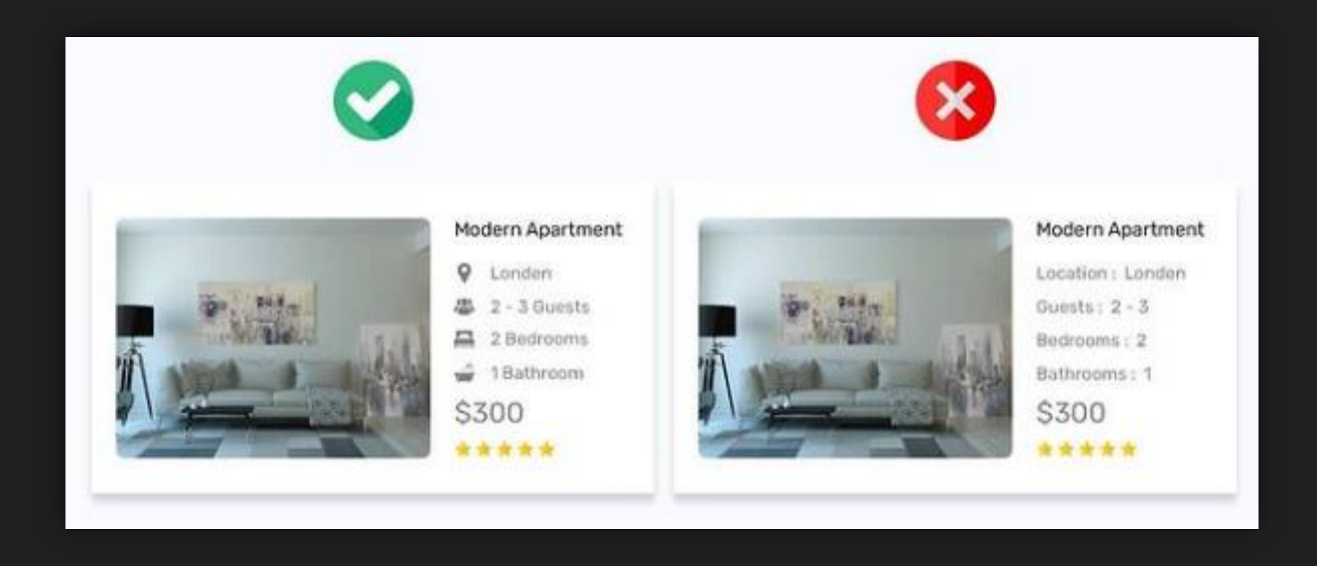
لا تكثر من التفاصيل، حتى لا تربك المستخدم، حتى لا يدعو عليك. بعض التفاصيل يمكن اختصارها، كالتفاصيل الثانوية التي تشير إلى التفاصيل الأولية. كما أن اختصار هذه التفاصيل المتكررة ليس فقط يزيد من وضوح التطبيق، وإنما أيضًا يزيد من جماليته.



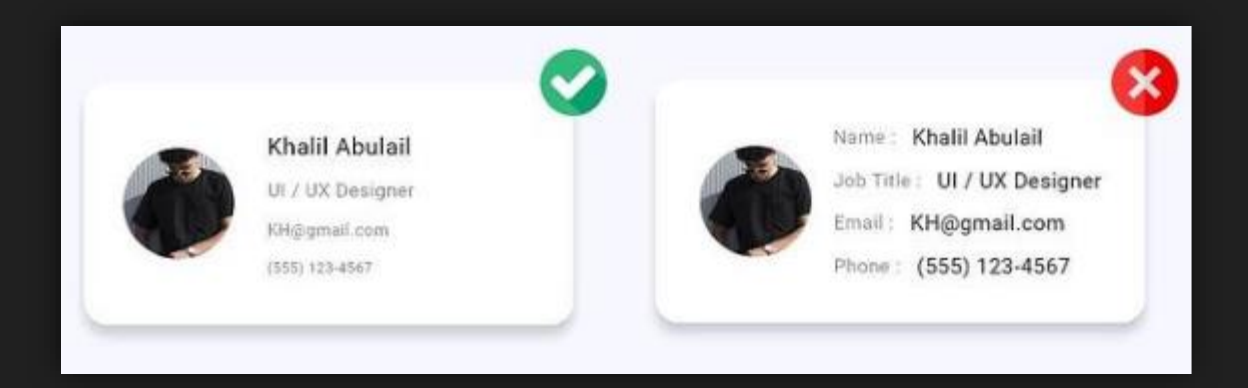
لكن بالمقابل، وجود بعض التفاصيل أحيانًا أفضل من قلتها:



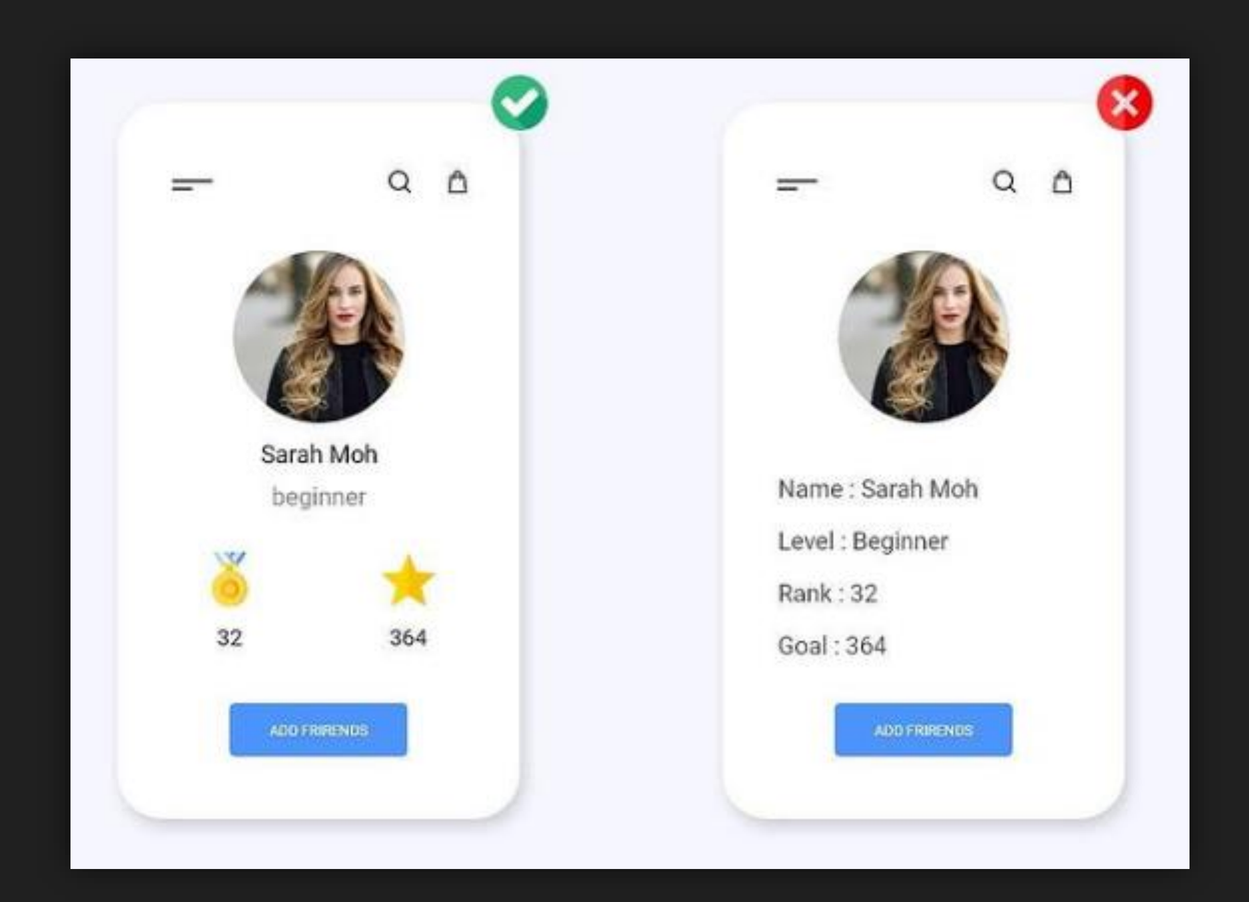
اختصر التفاصيل الشائعة (كمسميات الحقول) بأيقونات إن أمكنك:



بعض التفاصيل لا تحتاج إيضاحًا أصلًا، ووجودها مثل غيابها، بل إن وجودها أحيانًا ولو كأيقونات يؤثر سلبًا:



لاحظ الفرق بين التصميمين:



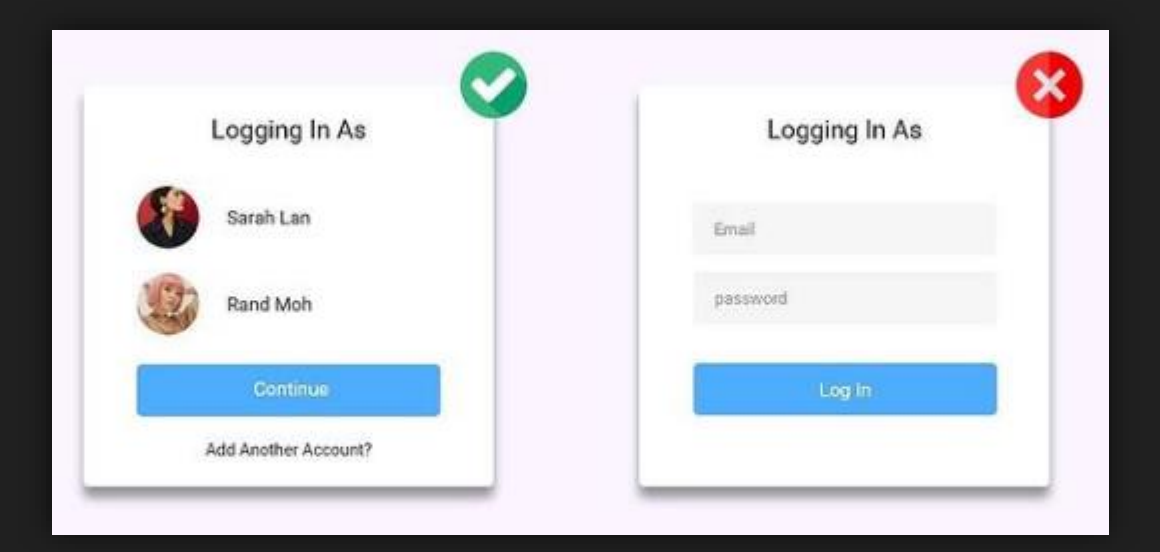
التصميم، للمستخدم أولًا وآخرًا

تجنب تصميم الحقول التي تحتوي على عمودين، قدر الإمكان.. فالمستخدم يقوم بفحص الصفحة أو النافذة بشكل شاقولي، خصوصًا في الأجهزة المحمولة. أما في نوافذ تطبيقات الحواسيب أو المواقع فالمستخدم يمسح المحتوى على شكل حرف Z إذا كان المحتوى لا يحوي صورًا ولا عناوين وإنما مجرد نص كصفحة جريدة (إذا كان المحتوى باللغة العربية فسيمسح المحتوى على شكل حرف Z معكوس)، ليتوقّف عشوائيّا عند كلمة تجذب انتباهه. أما لو كانت الصفحة تحوي صورًا وعناوين (وأشبه بصفحة الكتاب هذه) فإن المستخدم سيمسح المحتوى على شكل حرف F أو E (معكوسين للمحتوى العربي)، لذلك فاهتم بأدواتك التي ترغب أن يركز عليها المستخدم، فزر يقع في أيمن وأعلى الصفحة في نافذة عربية سيتلقى نقرات أكبر من زر يقع في يسارها وأعلاها، والذي بدوره سيتلقى نقرات أكثر من زر يقع في أسفل الشاشة، أما الأزرار الأقل نقرًا فستكون تلك الموضوعة بشكل عشوائي في منتصف النافذة، مالم نفعل شيئًا يحسن من وضعها. [1]

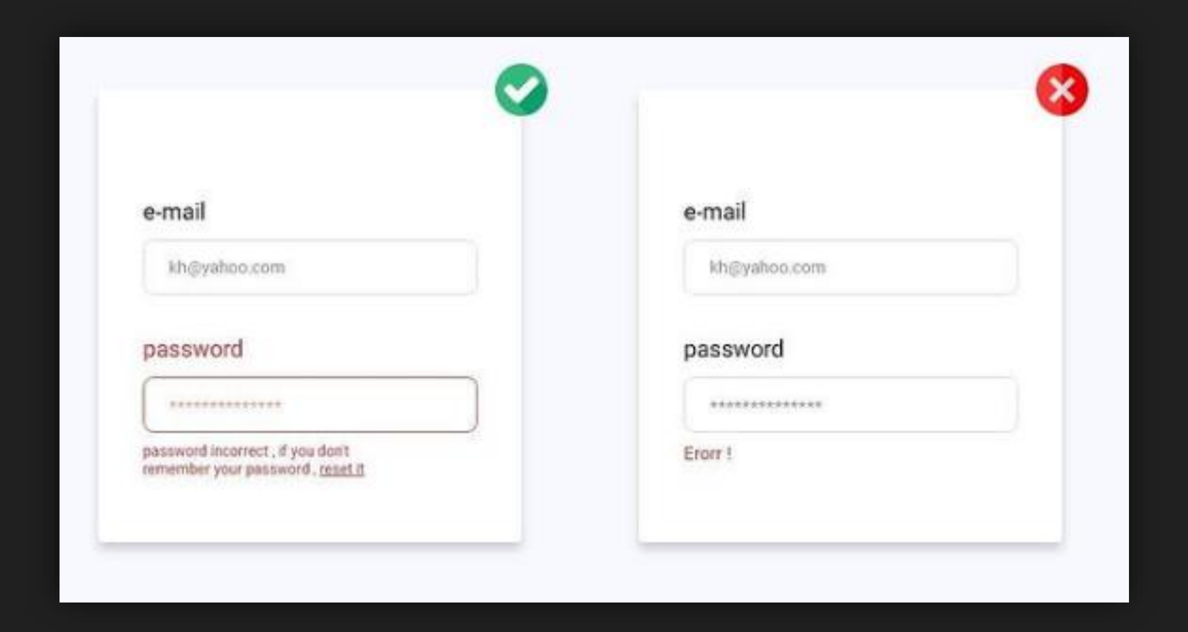
إذا كانت الحقول لا تتسع في صفحة واحدة جرب أن تفصلها لعدة خطوات منظمة تحت فروع. مثال: الخطوة الأولى: بيانات شخصية. الخطوة الثانية: معلومات التواصل. الخطوة الثالثة: معلومات الدفع. وهكذا...

حاول حفظ بعض المعلومات الأساسية عن المستخدم، خصوصًا إذا كان تطبيقك متعدد المستخدمين. لا تجبر المستخدم على إدخال كلمة المرور في كل مرة يغلق ويفتح فيها تطبيقك، واترك له الخيار في ذلك.

[1] انظر كتاب "مدخل إلى تجربة المستخدم" لـ محمد فواز عرابي، ص: 66-70. بتصرف.



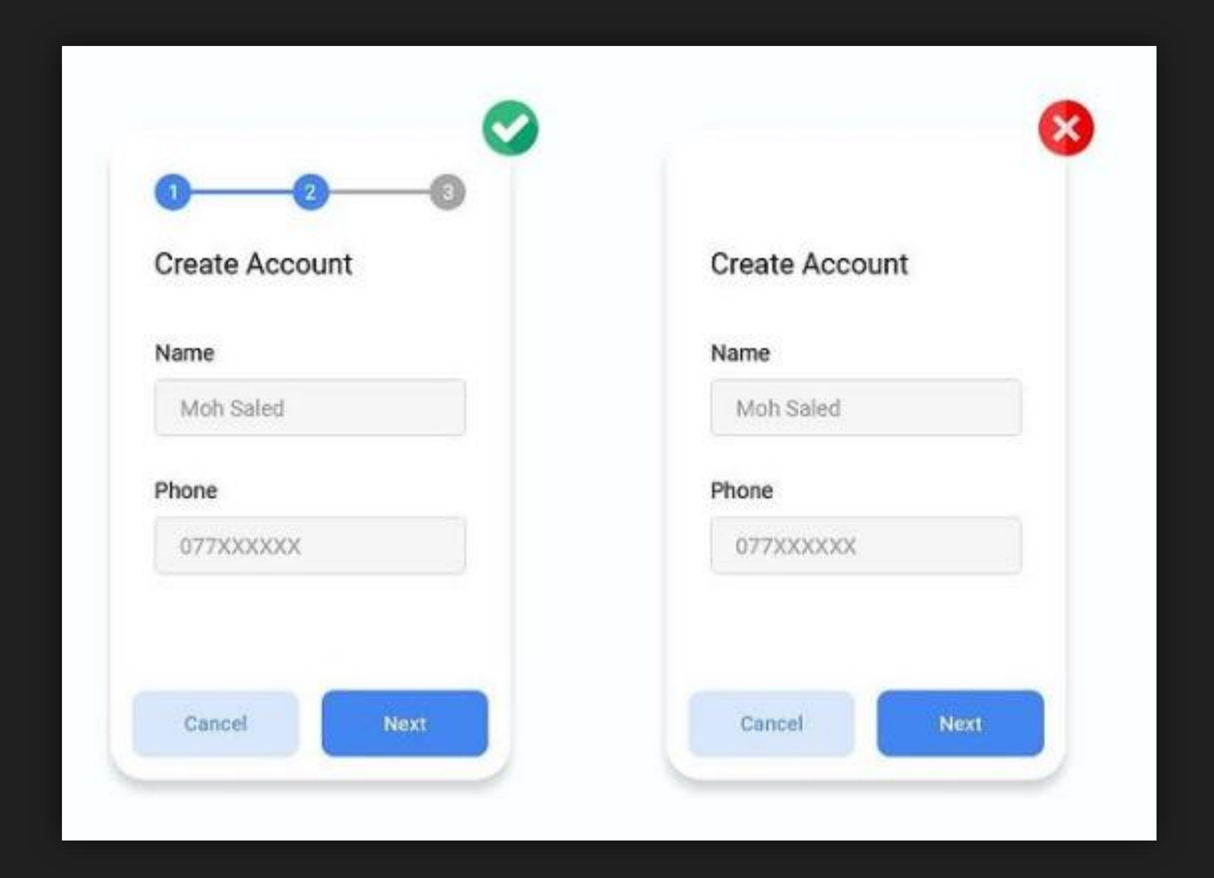
تجنب ترك المستخدم عالقًا في رسالة خطأ دون إيضاح كاف عن نوع الخطأ ومكانه بشكل صحيح. فمثلًا قد يدخل المستخدم كلمة سر خاطئة، عوضًا عن إرسال رسالة مفادها "حصل خطأ ما" [1]، أرسل رسالة تقول فيها للمستخدم بما معناه "بعد إذنك كلمة السر غير صحيحة". فأخطاء كهذه كفيلة بجعل المستخدم يترك تطبيقك أو موقعك، ويعطيك تقييمًا سيئًا، ويشهر بك لأصحابه ويضرب سمعتك.



[1] كتطبيق MyMTN Syria الذي يصر على إرسال رسالة عقيمة محتواها كلمتين فقط "حصل خطأ" عند تسجيل الدخول، مع أنني أدخلت كلمة السر بشكل صحيح وكنت مستعدًّا للحلفان عليها 🐸!

تجنب استخدام القائمة المنسدلة ComboBox لتحديد خيارات معدودة، فالقائمة المنسدلة تحتاج نقرتين لاختيار عنصر من القائمة. وعوضًا عن ذلك استخدم قائمة عادية ListBox أو أزرار اختيار RadioButton (قد تسمى OptionButton) والتي تحتاج نقرة واحدة لتغيير الاختيار. وبالمقابل تجنب استخدام القائمة العادية أو أزرار الاختيار لتحديد خيارات كثيرة. وفي حال كانت الخيارات أكثر من 25 خيار، فحاول إضافة حقل بحث يمكّن المستخدم من البحث عن خيار بكتابته، ولأدوات ComboBox إمكانية ربطها مع مصدر بيانات لتفعيل إمكانية البحث عن عناصر داخلها.

عند وجود خطوات محدودة في طريق المستخدم، يفضل إيضاحها. (سنتناول تصميم هذه الأداة في الفصل السابع)



كلمة أخيرة

في الفترة الأخيرة كثرت الصفحات والقنوات العربية على وسائل التواصل الاجتماعي التي تهتم بهذه المفاهيم، أذكر منها على سبيل المثال صفحة UX Itar [1] على فيسبوك وصفحة Khalil abulail [2] على إنستغرام.

وعلى سيرة صفحة Khalil abulail، بعض المعلومات في هذا الفصل مأخوذة منه (الصور التي فيها تصميمين، أحدهما صحيح والآخر خاطئ)، كما أني اعتمدت على صفحات أخرى مثل Zeroone Pro وبلورة ديزيان على فيسبوك.

كانت لي نية في إيجاز بعض مفاهيم المصطلحين في فصل منفصل، ولكنني تراجعت عن ذلك بعد إيجادي لكتاب عربي يشرح مفاهيم UX بشكل منمق وجميل، وهو كتاب باسم "مدخل إلى تجربة المستخدم (User Experience – UX)" لمحمد فواز عرابي. لذلك فقد اكتفيت بذكر بعض الفروقات بين المصطلحين ونبذة عن كل منهما، ولنا وقفة أخرى في الباب الثاني من هذا الكتاب مع هذين المصطلحين إن شاء الله.

[1] صفحة UX Itar على فيسبوك https://www.facebook.com/uxitar/.
[2] صفحة Khalil abulail على إنستغرام https://www.instagram.com/abulailkhalil/.

الفصل الثاني – أدوات تصميم تطبيقات ويندوز

يظن البعض أن التطبيقات Applications هي تلك التي تعمل على أنظمة الأندرويد فقط، أما ما يعمل على نظام ويندوز هو البرامج Programs، وهذا غير صحيح.

البرامج Programs هي مجموعة من التعليمات التي ينفذها الحاسوب، ومنها جاء مصطلح البرمجة. أما التطبيقات Applications فهي برنامج أو عدة برامج تساعد المستخدم على أداء مهام أو وظائف أو نشاطات معينة.

يمكن اعتبار أي كود برنامجًا، وبكلام كائني التوجه: كل فئة تعتبر برنامجًا، وبكلام أدق: كل طريقة Method هي برنامج. هذا إذا ما افترضنا نظافة أكوادك واستقلالية فئاتك (أن

تكون لكل فئة وظيفة – أو مجموعة وظائف – معينة، وألا تتداخل وظائف الفئات مع بعضها).

تعتمد التطبيقات على البرامج، إذ لا وجود للتطبيقات لولا وجود البرامج، بينما لا حاجة للبرامج لوجود التطبيقات. كما أن البرامج قد تكون ملفات عديدة، بينما لا تكون التطبيقات إلا ملفًّا واحدًا (هو الملف التنفيذي Exe في أنظمة ويندوز والتطبيق Apk في أنظمة الأندرويد).

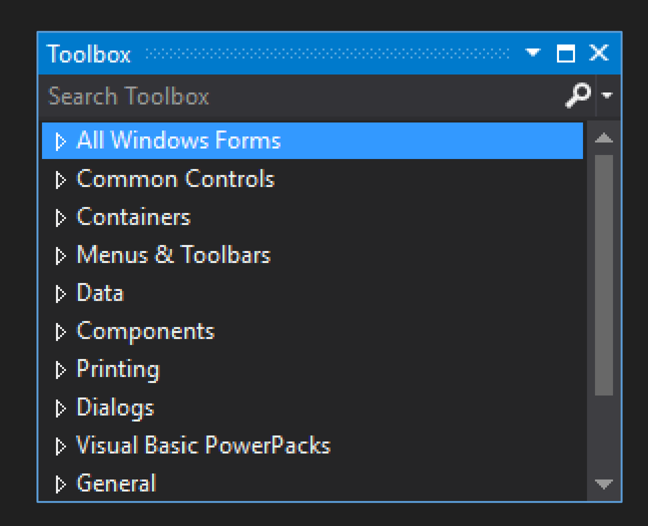
سنناقش في هذا الفصل الأدوات القياسية التي يقدمها لك الفيجوال ستوديو، وكيفية برمجتها، بالإضافة لتقنيات توثيق الفئات واستعراضها.

الأدوات القياسية في ويندوز

أدوات مايكروسوفت عامة، كلاسيكية، ومحافظة على نفسها خلال تاريخها، فالشركات الكبيرة يهمها أن تكون أدواتها أساس لأدوات من سواها. ومع ذلك، فإنك لن تجد في البرامج الاحترافية أثرًا لأدوات مايكروسوفت التقليدية، أو قد تجد عددًا محدودًا منها (كصناديق النصوص)، في حين أن أدوات هذه البرامج غالبًا ما تكون مصممة من قبل شركات أخرى مختصة بالتصميم، أو من تصميم شركات هذه البرامج نفسها.

من المعلوم أنه لابتكار تقنيات جديدة لا بدّ لك من مواكبة أحدث التقنيات المتوفرة أو التقنيات الأساسية والقياسية على أقل تقدير، وعليه فإنه ينبغي عليك أن تكونَ مطّلعًا وبشكل جيد على أغلب الأدوات البرمجية القياسية في ويندوز مثل صناديق النصوص وصناديق اللوائح والأزرار وغيرها، وذلك لتتمكن من إنشاءِ أدوات جديدة إما على مبدأ الأدوات القياسية أو الأدوات المتوفرة وذلك بإضافة ميزات جديدة تجذب المبرمجين لاستخدام أدواتك أو لتقضي لك حاجاتٍ لا تؤمّنها الأدوات المتوفرة، أو ابتكارِ أدوات جديدة كليًّا لم يُسبق لأحد أن أنشأها.

تقسم الأدوات القياسية في لغة #C إلى مجموعات كما هو موضّح بالصورة التالية، وهي كما يلي:



- جميع أدوات Windows Forms القياسية: وتحتوي على جميع الأدوات القياسية، المقسمة وفق الأقسام الأخرى.
- الأدوات الأكثر شيوعًا Common Controls: هي عبارة عن أكثر الأدوات استخدامًا في أي برنامج، إذ إنه تكاد لا تخلو أي نافذة من واحدة على الأقل من هذه الأدوات، كالأزرار وصناديق النصوص والعناوين Labels.
- الحاويات Containers: تستخدم كأُطُر وحاويات لاحتواء أدوات أخرى، فعندما تصبح أداة ما حاوية لأداة أو مجموعة من الأدوات الأخرى فإنها تصبح الأداة الأم لها.
- أشرطة الأدوات والقوائم Menu & Toolbars: هي عبارة عن صفوف شاقولية أو أفقية من الرموز (الأيقونات) القابلة للنقر والتي تؤدي وظائف نظام التشغيل أو تطبيق معين. أشرطة الأدوات شائعة الاستخدام في متصفحات الويب وتطبيقات معالجة النصوص وأنظمة التشغيل ومواقع الويب. وهي مصممة لتوفير وصول سهل وفوري إلى الوظائف الأكثر استخدامًا للمستخدمين.[1]
- البيانات Data: تعطيك هذه الأدوات إمكانية ربط بياناتك المختلفة – وخصوصًا المجمعة في قاعدة بيانات – مع أدوات عرض البيانات بأشكال عرض مختلفة كالمخططات Charts وجداول عرض البيانات DataGrid.

[1] راجع موقع Techopedia – ماهي أشرطة القوائم https://www.techopedia.com/definition/186/toolbar.

- المكونات Components: تسمح لك هذه الأدوات بالتعامل مع وظائف نظام التشغيل.
- الطابعات Printing: تستخدم لأعمال الطباعة 🐸.
- صناديق الحوار Dialogs: تعطيك إمكانية التعامل مع مربعات حوار اختيار الخطوط والألوان وغيرها.
- أدوات الرسم الخاصة بـ VB: تمكّنك من رسم أشكال هندسية بسيطة كالخطوط والمستطيلات وغيرها.
- أدوات عامة General: أدوات أخرى (غير قياسية، تضيفها أنت).

وما سنتناوله في هذه الفقرة هو أكثر الأدوات شيوعًا وبضعةُ أدوات أخرى، لنبني عليها أدواتنا التي سنصممها في الباب الثالث إن شاء الله.

تعتبر الأدوات – الأزرار وصناديق الصور والاختيار و ... – كائنات برمجية، فهي بالأصل فئات (وكما تعلم فالفئة هي قالب عام لشيء برمجي ما، ومنها ننشئ الكائنات) ومن المعروف أن لكل فئة مجموعة من الخصائص Properties ومجموعة من الطرق Methods ومجموعة من الأحداث Events. من الضروري أن تكون على اطلاع على قائمة بأكثر الخصائص والأحداث شيوعًا قبل أن تبدأ بتصميم أدوات جديدة، وقبل هذا وذاك، سنستعرض الأدوات نفسها، مع بعض الأمثلة التطبيقية.

الخصائص هي التي تحدد شكل وسلوك الأدوات، والأحداث هي التي ستعطيك إشعارًا بكل ما يمر على تطبيقك وأدواته، أما الطرق فهي ما يمكن للأداة أن تفعله، ولكل أداة حدث افتراضيّ تقوم به [1]، وعلى اعتبار أن الأحداث هي إجراءات [2] فلها وسطاء، وعادةً ما يكون لكل حدث وسيطين، الأول sender يمثل نسخة عن الكائن الذي رفع الحدث [3]، ويمكنك الاستفادة من هذا الوسيط في الحصول على معلومات عن الأداة التي رفعت الحدث عند التعامل مع أكثر من أداة في ذات الحدث. والثاني e والذي يمثل نوع الحدث

[1] عند النقر على أي أداة مرتين أثناء تصميم البرنامج تنتقل لحدث ما، هذا هو الحدث الافتراضي.
[2] انظر كتابنا "C# من البداية حتى الإتقان" ص207.
[3] رَفْعُ الحدث: تنفيذه. وقد يقال: تفجير الحدث.

وفيه معلومات عن الحدث. ففي الحدث MouseDown مثلا يمكنك الحصول على زر الفأرة المضغوط وموقع المؤشر وغيرها من المعلومات من خلال هذا الوسيط.



في بداية كل أداة من الأدوات التي سأشرحها في هذه الفقرة سأبدأ بالشرح الذي قدّمَتْه ميكروسوفت للمستخدمين في أدواتها، ثم سأشرح هذه الأدوات بشيء من الإيجاز.

المؤشر Pointer

وهو أول أداة من الأدوات الشائعة، وله الرمز ، الفكرة منه ببساطة هي ضبط الأداة الحالية الموجودة في متناول اليد على لا شيء (عدم وجود أداة جاهزة لتصميمها)، وهو الوضع الافتراضي، وفيه يمكنك تحديد الأدوات الموجودة في النموذج Form للتعامل معها.

زر الأوامر Button

يَرفعُ حدثًا عندما ينقر المستخدم عليه، وله الرمز ، ويسمى في لغات أخرى CommandButton، وهو من أكثر الأدوات المستخدمة في البرامج، لدرجة أن أي نافذة من البرنامج تكاد لا تخلو من زر أو اثنين على الأقل!

لزر الأوامر شأن كبير في برامجك ليس فقط لاستخداماته، وإنما لإعطاء المظهر العام لبرنامجك، فالفصل السابع مثلًا سيعطيك أساليب رسم وتصميم زر الأوامر ليأخذ أشكالًا مختلفة كالدائرية أو التي على شكل قطع ناقص [1] أو أي شكل كان.

Start

[1] قطع ناقص: شكل إهليلجي، يشبه الدائرة إلا أن له قطران (للدائرة قطر واحد).

صندوق الاختيار CheckBox

يسمح للمستخدم باختيار أو مسح خيار محدد، وله الرمز ☑، ويستخدم عند وجود خيارات معينة يمكن تفعيلها أو إلغاءها (بحيث يمكن اختيار أكثر من خيار في الوقت نفسه).

☑ I agree to the terms of use and Privacy statements

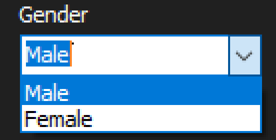
صندوق لائحة قابلة للاختيار CheckedListBox

يُظهر قائمة بمجموعة من العناصر ومعها صندوق اختيار على يسار كل عنصر، وله الرمز ، ويستخدم لعرض مجموعة من العناصر على شكل قائمة، بحيث يمكن اختيار مكوناتها.

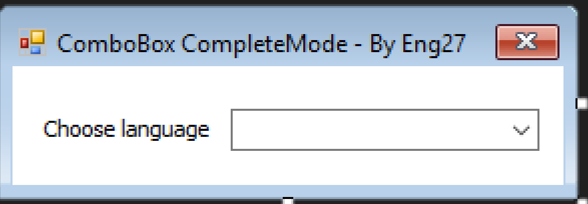
Select what you want to download:
☑ Primary files
☑ Documentations
☐ Additional assets

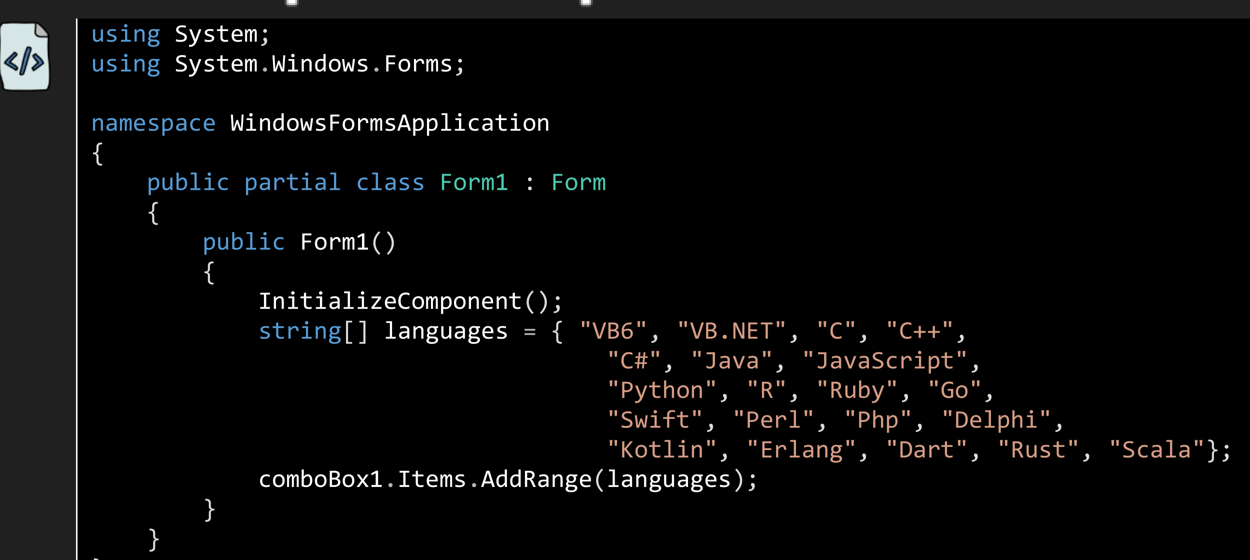
صندوق اللائحة المنسدلة ComboBox

يُظهر صندوق نص قابل للتعديل ومعه قائمة منسدلة بقيم محددة، وله الرمز ، ويستخدم لعرض مجموعة من العناصر على شكل قائمة منسدلة، بحيث لا يظهر إلا العنصر المختار، وبقية العناصر مخفية.



يمكنك ضبط وضع الإكمال التلقائي – مثل عمليات البحث في غوغل، تجد النص يكتمل عند كتابة بعض الحروف منه – عبر مجموعة من الخصائص، صمم النافذة التالية لنجري عليها بعض التجارب:





```
using System;
using System.Windows.Forms;

namespace WindowsFormsApplication
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            string[] languages = { "VB6", "VB.NET", "C", "C++",
                                   "C#", "Java", "JavaScript",
                                   "Python", "R", "Ruby", "Go",
                                   "Swift", "Perl", "Php", "Delphi",
                                   "Kotlin", "Erlang", "Dart", "Rust", "Scala"};
            comboBox1.Items.AddRange(languages);
        }
    }
}
```

الخصائص التي ستضبط وضع الإكمال التلقائي:

AutoCompleteCustomSource
AutoCompleteMode
AutoCompleteSource

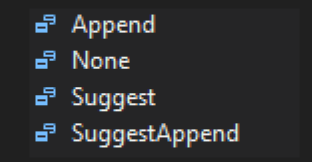
لنبدأ بمصدر الإكمال التلقائي AutoCompleteSource، يمكن أن يأخذ أحد القيم التالية:

- AllSystemSources
- AllUrl
- CustomSource
- FileSystem
- FileSystemDirectories
- HistoryList
- ListItems
- None
- RecentlyUsedList

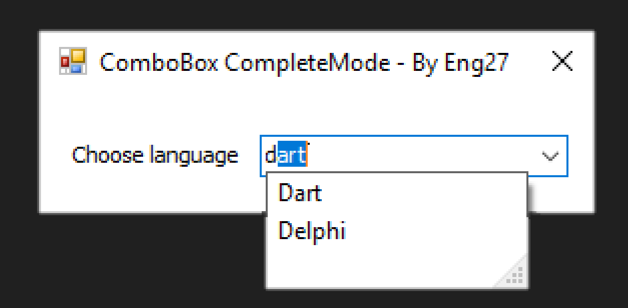
إذا أردت أن يتم إكمال النتيجة من خلال محتوى الأداة نفسها اختر ListItems، مثل حالة المثال المطروح، نرغب من البرنامج أن يكمل النص إذا كان موجودًا ضمن القائمة. أما إذا

أردت إضافة مجموعة من العناصر لا علاقة لها بمحتوى الأداة فيمكنك ذلك من خلال CustomSource، مثل حالة التعامل مع قواعد البيانات (لديك مجموعة من العناصر تتغير بتغير بيانات قاعدة البيانات، والتي ترغب أن يتم الإكمال بحسب هذه البيانات، فإنك تحول العناصر إلى مصفوفة نصية ثم تسندها لخاصية مصدر الإكمال التلقائي الخاص (AutoCompleteCustomSource أو ببساطة مثل حالة إدخال عناصر الإكمال التلقائي يدويًّا إلى الخاصية المذكورة.

أما بالنسبة لوضع الإكمال التلقائي AutoCompleteMode فيأخذ إحدى القيم التالية:

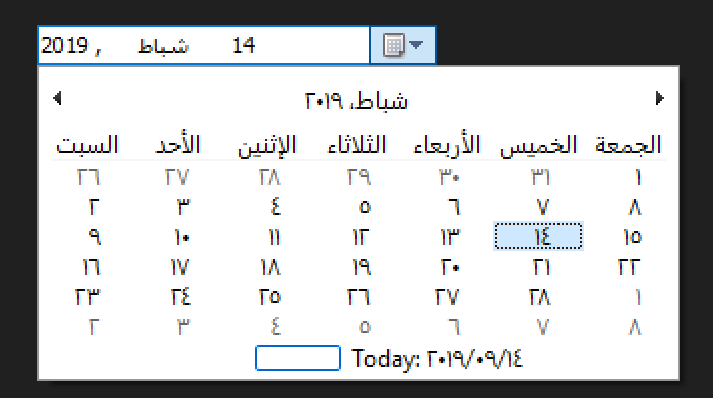


إذا أردت عرض اقتراحات فقط اختر Suggest، أما إذا أردت الإكمال مباشرةً فاختر Append (في هذا الوضع يتم الإكمال على أساس أول عنصر مطابق)، ولتحصل على أفضل أداء فاضبط الخاصية على SuggestAppend ليتم إكمال النص وعرض الاقتراحات الموافقة.



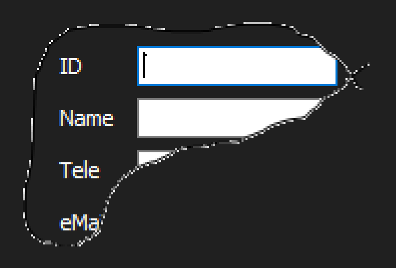
أداة انتقاء التاريخ والوقت DateTimePicker

تسمح للمستخدم باختيار التاريخ والوقت، وعرضهما بفترة معينة، ولها الرمز .



أداة العنوان Label

تُزود برنامجك بمعلومات أثناء التنفيذ run-time، أو نص توضيحي لأداة ما، ولها الرمز A، وهي أشبه بالكتابة على الجدران، فتسمح لك بالكتابة حيثما شئت في نافذة تطبيقك.



أداة العنوان التشعبي LinkLabel

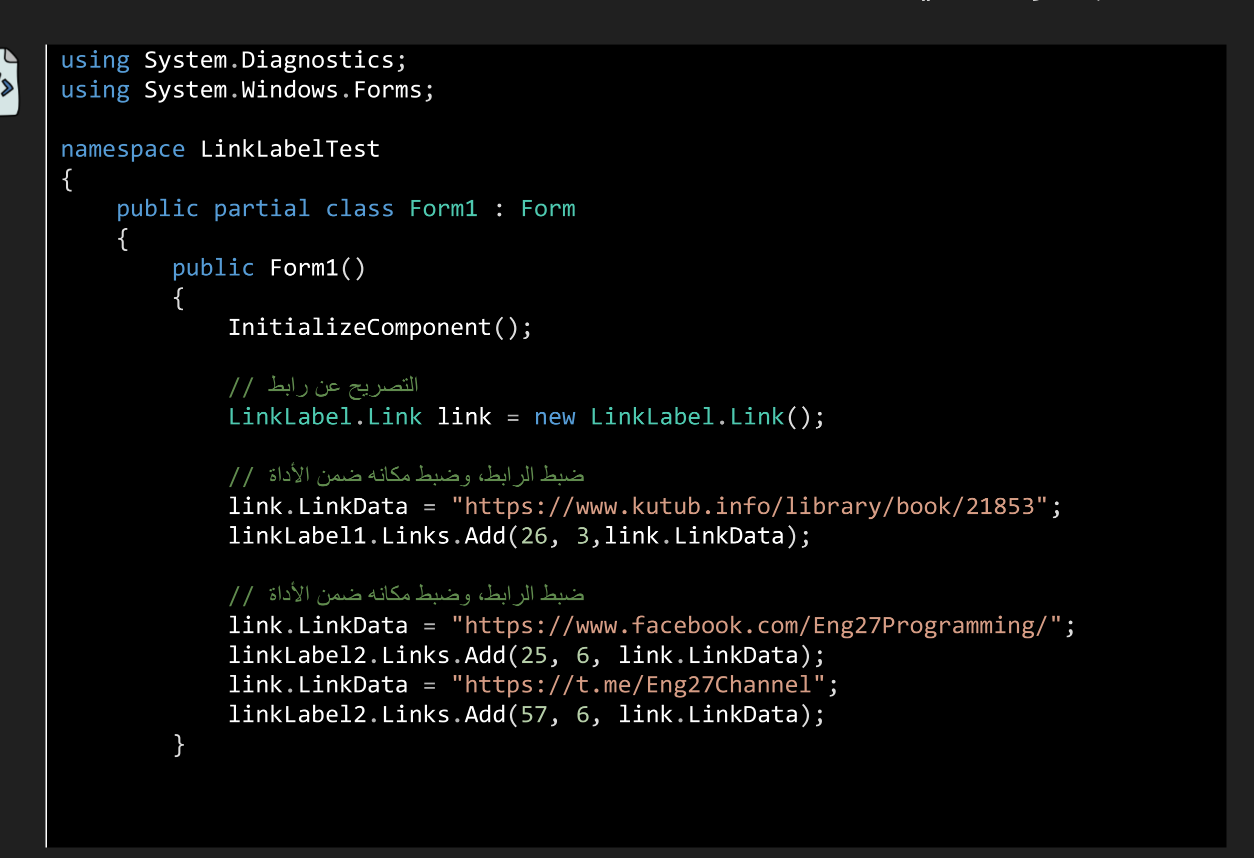
تعرض أداة عنوان تدعم وظائف الروابط التشعبية، وتنسيقها، والتعامل معها، ولها الرمز A، وتستخدم لربط تطبيقك بالروابط الإلكترونية الخارجية أو ببرامج ووظائف معينة.

وكمثال عليها، تجد كثيرًا من المواقع فيها عبارة نصية محتواها "لتحميل الملف اضغط هنا" مثلًا، ويمكن إنشاء مثل هذه العبارة في تطبيقات ويندوز باستخدام أداة LinkLabel، فهي ليست كما يعتقد البعض – أن قيمتها النصية هي بالكامل الرابط المطلوب – وإنما يمكن تضمين الرابط ضمن كلمة أو جملة كما تلاحظ في المواقع وصفحات الويب.

أضف أداتَي LinkLabel إلى مشروع نوافذ، وغير خاصية النص في كل منهما إلى ما هو واضح بالصورة التالية:



استخدم الكود التالي:



```
using System.Diagnostics;
using System.Windows.Forms;

namespace LinkLabelTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            // التصريح عن رابط
            LinkLabel.Link link = new LinkLabel.Link();

            // ضبط الرابط، وضبط مكانه ضمن الأداة
            link.LinkData = "https://www.kutub.info/library/book/21853";
            linkLabel1.Links.Add(26, 3,link.LinkData);

            // ضبط الرابط، وضبط مكانه ضمن الأداة
            link.LinkData = "https://www.facebook.com/Eng27Programming/";
            linkLabel2.Links.Add(25, 6, link.LinkData);
            link.LinkData = "https://t.me/Eng27Channel";
            linkLabel2.Links.Add(57, 6, link.LinkData);
        }
```

```
        // الانتقال للرابط
        private void linkLabel1_LinkClicked
             (object sender, LinkLabelLinkClickedEventArgs e)
        {
            Process.Start(e.Link.LinkData as string);
        }

        private void linkLabel2_LinkClicked
              (object sender, LinkLabelLinkClickedEventArgs e) { }
    }
}
```

لاحظ الروابط بعد تنفيذ البرنامج:

يمكنك تحميل هذا الكتاب من هنا.

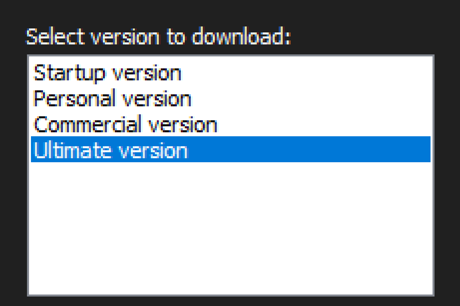
يمكنك الانضمام إلينا على فسبوك، والاشتراك في قناتنا على تلغرام أيضا.

تعمدت أن يحتوي المثال على أداتين من LinkLabel لتلاحظ إمكانية وضع أكثر من رابط ضمن الأداة نفسها.

يمكنك أيضًا وضع مسارات ملفات أو مجلدات موجودة ضمن القرص الصلب كرابط للانتقال إليه، إذ إن الطريقة Process.Start تعطيك إمكانية تشغيل البرامج أو فتح المجلدات.

صندوق اللائحة ListBox

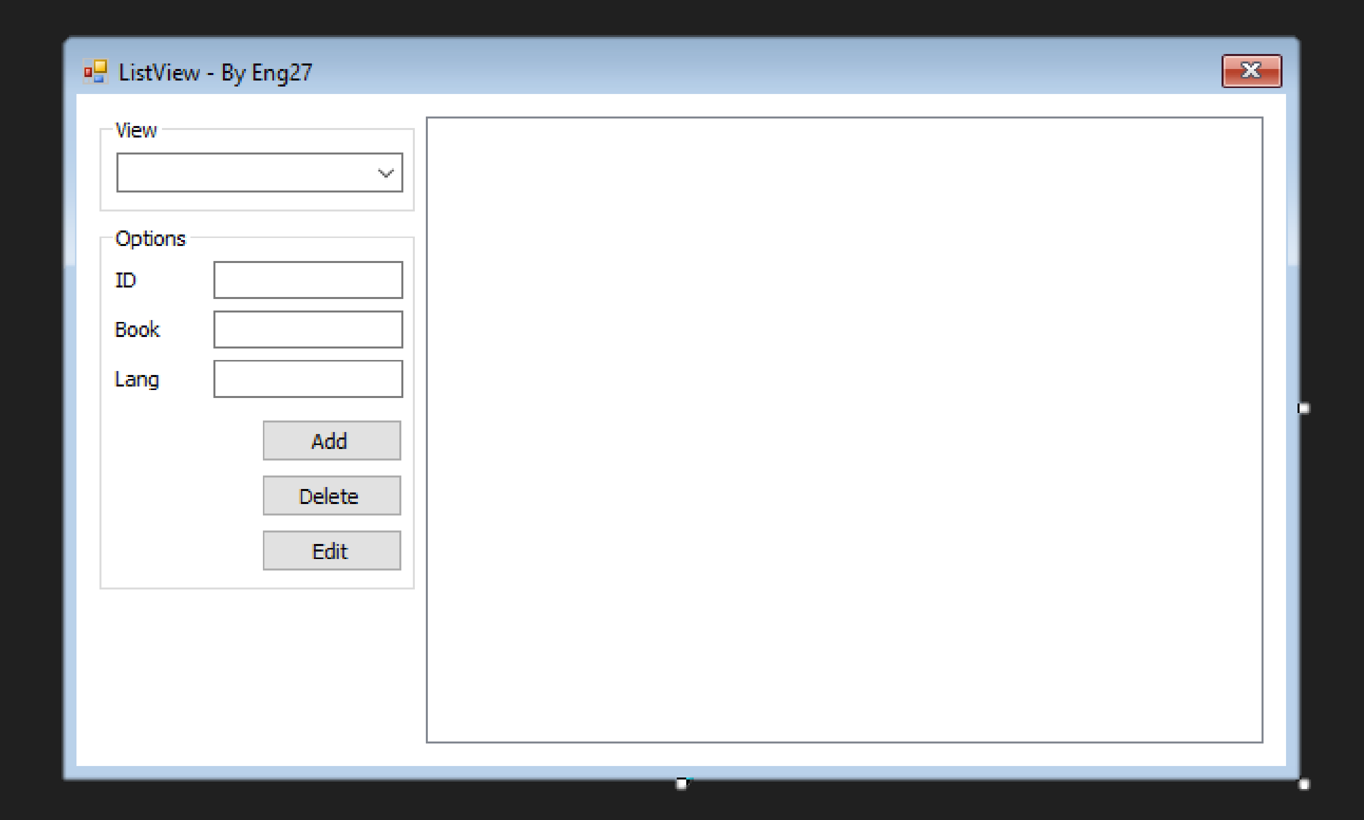
يعرض قائمة بحيث يمكن للمستخدم اختيار عناصر منها، وله الرمز .



قائمة العرض ListView

تعرض العناصر بواحدة من خمسة أشكال، ولها الرمز ، وغالبًا ما تستخدم مع الملفات، أو لتمثيل الجداول، البسيطة منها خصوصًا.

صمم النافذة التالية لإيضاح بعض ما يمكنك القيام به من خلال هذه الأداة:



ثم استخدم الكود:

```
using System;
using System.Windows.Forms;

namespace WindowsFormsApplication
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            string[] lvTypes = { "LargeIcon", "Details",
                                 "SmallIcon", "List",
                                 "Tile" };

            comboBox1.Items.AddRange(lvTypes);
            comboBox1.Text = "Details";

            listView1.Columns.Add("ID");
            listView1.Columns.Add("Book").Width = 150;
            listView1.Columns.Add("Language");

            listView1.FullRowSelect = true;
            listView1.MultiSelect = false;
        }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            if (textBox1.Text != "" & textBox2.Text != "" & textBox3.Text != "")
            {
                ListViewItem L;
                L = listView1.Items.Add(textBox1.Text);
                L.SubItems.Add(textBox2.Text);
                L.SubItems.Add(textBox3.Text);
                textBox1.Clear();
                textBox2.Clear();
                textBox3.Clear();
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            if (listView1.SelectedItems.Count > 0)
            {
                listView1.SelectedItems[0].Remove();
                textBox1.Clear();
                textBox2.Clear();
                textBox3.Clear();
            }
        }

        private void button3_Click(object sender, EventArgs e)
        {
            if (listView1.SelectedItems.Count > 0)
            {
                listView1.SelectedItems[0].Text = textBox1.Text;
                listView1.SelectedItems[0].SubItems[1].Text = textBox2.Text;
                listView1.SelectedItems[0].SubItems[2].Text = textBox3.Text;
                listView1.SelectedItems.Clear();
            }
        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            listView1.View = (View)Enum.Parse(typeof(View), comboBox1.Text);
        }

        private void listView1_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (listView1.SelectedItems.Count > 0)
            {
                textBox1.Text = listView1.SelectedItems[0].Text;
                textBox2.Text = listView1.SelectedItems[0].SubItems[1].Text;
                textBox3.Text = listView1.SelectedItems[0].SubItems[2].Text;
            }
        }
    }
}
```

جرب البرنامج، أضف واحذف وعدّل البيانات، حاول فهم الأسطر السابقة واربطها بما حصل أمامك عند تجريب البرنامج، الأسطر التالية قد تساعدك على فهم مجريات الأمور:

- في البداية لدينا التابع البنّاء للنافذة، والذي يهيئ في البداية مكونات البرنامج ثم يعرّف مصفوفة خطية فيها إمكانيات العرض الخاصة بقائمة العرض، ثم يضيف عناصر هذه المصفوفة لصندوق اللائحة المنسدلة ComboBox ويختار وضع التفاصيل، ثم يضيف بعض القوائم ويضبط الخاصية التي تمكن المستخدم من اختيار العناصر على شكل صفوف عند اختيار وضع العرض على شكل تفاصيل، ويضبط خاصية أخرى تمنع المستخدم من اختيار أكثر من سطر بالوقت نفسه.
- حدث النقر على الزر الأول يقوم بإضافة عنصر بتفاصيل ثانوية في حال لم تكن صناديق النصوص فارغة، ثم يُفرغ محتوى صناديق النصوص.
- حدث النقر على الزر الثاني يقوم بحذف العنصر المحدد ويُفرِغ محتوى صناديق النصوص في حال كان هناك عناصر محددة في قائمة العرض.
- حدث النقر على الزر الثالث يعدل العنصر المحدد في قائمة العرض ليساوي محتوى صناديق النصوص في حال كان هناك عناصر محددة، ويلغي تحديد العناصر من قائمة العرض.
- حدث تغيير دليل العنصر المحدد في صندوق اللائحة المنسدلة يغير وضع عرض قائمة العرض إلى ما هو محدد في صندوق اللائحة المنسدلة، وعلى اعتبار أن القيمة المحددة في هذا الصندوق نصية string فينبغي تحويلها إلى Enum من النوع View.
- حدث تغيير دليل العنصر المحدد في قائمة العرض يؤدي إلى تغيير محتوى صناديق النصوص إلى محتوى العنصر المحدد، للدلالة على العنصر المحدد بشكل أكبر.



في الفصل الثامن ستجد فئة تحول القيم النصية إلى قيمة من نوع المعددات (من النوع Enum).

صندوق النص المُقَنَّع MaskedTextBox

يستخدم قالب جاهز معين للتمييز بين صيغة مقبولة للنص وأخرى غير مقبولة له، وله الرمز .

يمكنك استخدام إحدى هذه الأقنعة:

Input Mask ? X

Select a predefined mask description from the list below or select Custom to define a custom mask.

Mask Description	Data Format	Validating Type
Phone Number	(012)345-6789	(none)
Phone Number no Area C...	123-4567	(none)
Short Date	26 /10 /2005	DateTime
Short Date/Time	26 /10 /2005 14:30	DateTime
Social Security Number	123-45-6789	(none)
Time	2:30	DateTime
Time (24 Hour)	14:30	DateTime
<Custom>		(none)

Mask: ☑ Use ValidatingType

Preview:

OK Cancel

الجدول التالي يوضّح أسس إنشاء قناع خاص:

الرمز	المعنى
0	يقبل أرقامَ نظام العد العشري [1] فقط.
9	يقبل أرقام نظام العد العشري والفراغات.
#	يقبل أرقام نظام العد العشري والفراغات أو + أو -.
L	يقبل الأحرف الكبيرة والصغيرة.
C أو &	محرف (حرف أو رمز أو رقم).
A أو a	أرقام أو أحرف.
\\	يقبل الرمز \.

[1] نظام العد العشري: الأرقام من 0 وحتى 9.

فمثلا: لو جعلت القناع بالشكل "0LLL" فإنه سيقبل محتوى نصي مكون من أربعة محارف فقط، الأول رقم حصرًا والبقية أحرف حصرًا.

أيقونة الإشعارات NotifyIcon

تعرض أيقونة في منطقة الإشعارات، في الزاوية السفلى اليمينية (بجانب الساعة)، أثناء عمل البرنامج run-time، ولها الرمز ، والتي تمكن المستخدم من التعامل مع تطبيقك حتى لو لم تكن نافذته ظاهرة على الشاشة.

صندوق النص الرقمي NumiricUpDown

يعرض قيمة رقمية والتي يمكن للمستخدم زيادتها أو إنقاصها من خلال زرين أحدهما للزيادة والآخر للإنقاص، وله الرمز ، ويمكنك الاستفادة منه عند حاجتك لصندوق نص يتعامل مع الأعداد فقط، مثل الأموال والحسابات وغيرها.

صندوق الصورة PictureBox

يعرض صورة، وله الرمز .

شريط التقدم ProgressBar

يعرض شريطاً، والذي يمتلئ ليشير للمستخدم بتقدم عملية ما، وله الرمز .

زر الراديو RadioButton

يسمح للمستخدم باختيار خيار وحيد من مجموعة خيارات عند دمجه مع مجموعة من أزرار الراديو، وله الرمز ، وهو على عكس صناديق الاختيار لا يسمح إلا باختيار واحد.

صندوق النص الغني RichTextBox

يقدم إمكانية إدخال نصوص ومزايا تعديل متقدمة مثل تنسيق المحارف والفقرات، وله الرمز ، وهو نافذتك نحو محررات النصوص.

صندوق النص TextBox

يمكّن المستخدم من إدخال النصوص، ويزوده بإمكانية تحرير النصوص على أسطر متعددة وقولبة محارف كلمات السر على شكل دائرة سوداء • ، وله الرمز ، وهو ثاني أشهر أداة بعد أداة الزر.

أداة التلميح ToolTip

تعرض معلومات عندما يحرك المستخدم المؤشر على أداة معينة، ولها الرمز ، ووجوده في تطبيقك مثل وجود البهار في الطعام، نكهة تجعله أشهى.

شجرة العرض TreeView

تعرض مجموعة هرمية من العناصر المسماة والتي يمكن لها أن تحتوي صورًا بشكل اختياري، ولها الرمز ، وتستخدم لعرض المكونات وفق تسلسل معين، وعلاقتها مع بعضها. من الأمثلة عليها أدوات تصفح الملفات، ولمثال جيّد عن هذا الموضوع أُحيلك إلى الرابط التالي: https://www.c-sharpcorner.com/article/display-sub-directories-and-files-in-treeview/.

متصفح الويب WebBrowser

يمكّن المستخدم من تصفح صفحات الويب من داخل تطبيقك، وله الرمز .

هذه كلها أدوات قياسية شائعة الوجود في مشاريع ويندوز، وسأوجز الأدوات الحاوية Containers في الأسطر القليلة التالية:

لوحة ذات نسق تدفقي FlowLayoutPanel

تتحكم بنسق[1] مكوناتها وترتبها تلقائيا بشكل تدفقي، ولها الرمز ، وبصراحة لم أجد ترجمة أفضل من هذه، عمومًا إذا أردت إنشاء تطبيقات تعرض بيانات وفق قوالب معينة (شبيهة بمنشورات الفيسبوك على سبيل المثال)؛ فاعتمد على هذه الأداة.

[1] نسق: مظهر، شكل، بنية.

صندوق التجميع GroupBox

يُظهر إطارًا حول مجموعة من الأدوات مع اسم اختياري، وله الرمز .

لوحة Panel

تسمح لك بتجميع الأدوات، ولها الرمز ، وبالنسبة لي تعتبر أحد أهم اختراعات البشرية في مجال التصميم على الإطلاق! (تشبه مفهوم الطبقات Layers في برامج التصميم الرسومية).

فصل الحاويات SplitContainer

تقسم منطقة عرض الحاوية – مثل اللوحات Panels – إلى منطقتين قابلتين لتغيير حجميهما من خلال فاصل Splitter، بحيث يمكنك إضافة أدوات فيهما بشكل منفصل عن بعضهما، ولها الرمز ، ويمكنك بها تنسيق نوافذ برنامجك بشكل محترم.

أداة التبويب TabControl

تدير وتعرض للمستخدم تبويبات متعلقة ببعضها والتي تحتوي الأدوات والمكونات المختلفة في برنامجك، ولها الرمز .

هذا ولا يسعنا إفراد وذكر كل شاردة وواردة من أدوات ويندوز، وباكتسابك للخبرة مع مرور الأكواد ستفهم الغاية من كل أداة بقراءة اسمها فقط، وقد تتنبأ بكيفية استخدامها حتى! وضع في ذهنك أن اكتفائي بهذا القدر من الأدوات ليس معناه أن هذا القدر سيكفي برامجك، إذ إن هناك مجموعة كبيرة من الأدوات التي يجب عليك الاطلاع عليها واختبارها وفهمها حتى لو لم نذكرها ونفصلها هنا، نذكر منها:

أدوات القوائم وأشرطة الأدوات Menus & Toolbars مثل ContextMenuStrip التي تعرض قائمة عند النقر بالزر الأيمن تمثل مجموعة من الوظائف المتعلقة بأداة معينة، أو MenuStrip التي تعرض شريطًا من القوائم أعلى البرنامج يمثل وظائف ومزايا البرنامج المختلفة، أو StatusStrip التي تعرض مجموعة من المعلومات – وبعض الوظائف أحيانًا –

أسفل نافذة البرنامج، أو ToolStrip التي تعرض مجموعة من الأدوات التي تختارها أنت كمبرمج لأداء وظائف معينة.

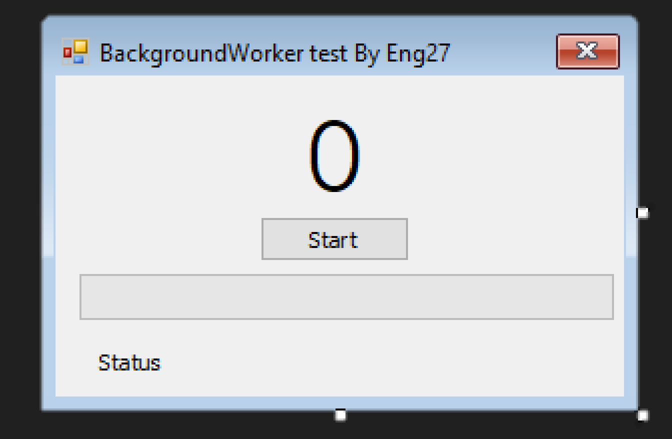
أدوات البيانات Data مثل الأداة Chart والتي تعرض المخططات والرسوم البيانية والتوضيحية والإحصائية، أو الأداة DataGridView التي تعرض جداول بيانات ذات وظائف قوية ومفيدة، أو الأداة DataSet والتي تربط برنامجك بقواعد البيانات المختلفة.

أدوات المكونات Components مثل ImageList والتي تخزن مجموعة من الصور في ذاكرة البرنامج ليتم عرضها في أدوات العرض المختلفة مثل شجرة العرض TreeView أو لائحة العرض ListView، أو Process والتي تمكنك بالاتصال بوظائف وتطبيقات نظام التشغيل المختلفة، أو Timer الذي يمكنك من أتمتة وظائف برنامجك أو تكرارها كل فترة. في الفقرات الفرعية التالية سأتناول بعض أدوات المكونات التي يُتوقع أنك لا تعرفها، في حين أن إضافتها لبرامجك ستضيف عليه لمسة إتقان أنت بحاجة لها:

العامل الخفي BackgroundWorker

من أكثر الأدوات التي تعجبني من أدوات المكونات هي الأداة BackgroundWorker، والتي تسمح لك بتنفيذ بعض الأكواد بالخلفية أثناء عمل البرنامج، والمثال التالي أنقله حرفيًّا من دورة حسونة أكاديمي لبساطته ووضوحه. لنفترض أننا نريد أن نغير قيمة أداة عنوان label معينة من 1 وحتى 100 ألف، لو استخدمنا حلقة for لفقدنا السيطرة على برنامجنا طوال فترة تنفيذ الحلقة، والحل هو تنفيذ الحلقة في الخلفية، وذلك كالتالي:

صمم النافذة المجاورة، واضبط شريط التقدم على قيمة أعظمية 100 ألف، ثم أضف أداة BackgroundWorker وغير خاصية WorkerReportsProgress لها لـ True، ثم استخدم الكود التالي:





```
using System;
using System.ComponentModel;
using System.Windows.Forms;

namespace BackgroundWorker
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); }

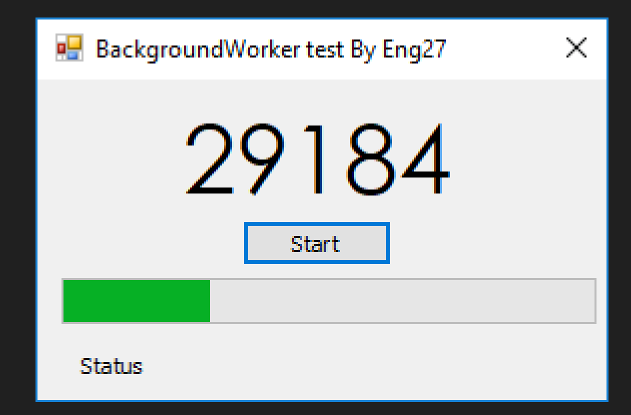
        private void button1_Click(object sender, EventArgs e)
        {
            if (backgroundWorker1.IsBusy) label2.Text = "Status: Is working!";
            else backgroundWorker1.RunWorkerAsync();
        }

        private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
        {
            for (int i = 1; i <= 100000; i++)
            {
                label1.Invoke((MethodInvoker)delegate
                {
                    label1.Text = i.ToString();
                });
                backgroundWorker1.ReportProgress(i);
            }
        }

        private void backgroundWorker1_ProgressChanged
             (object sender, ProgressChangedEventArgs e)
        {
            progressBar1.Value = e.ProgressPercentage;
        }

        private void backgroundWorker1_RunWorkerCompleted
              (object sender, RunWorkerCompletedEventArgs e)
        {
            label2.Text = "Status: Completed successfully!!";
        }
    }
}
```

هذه اللقطة أُخذت عشوائيًّا أثناء تنفيذ البرنامج:



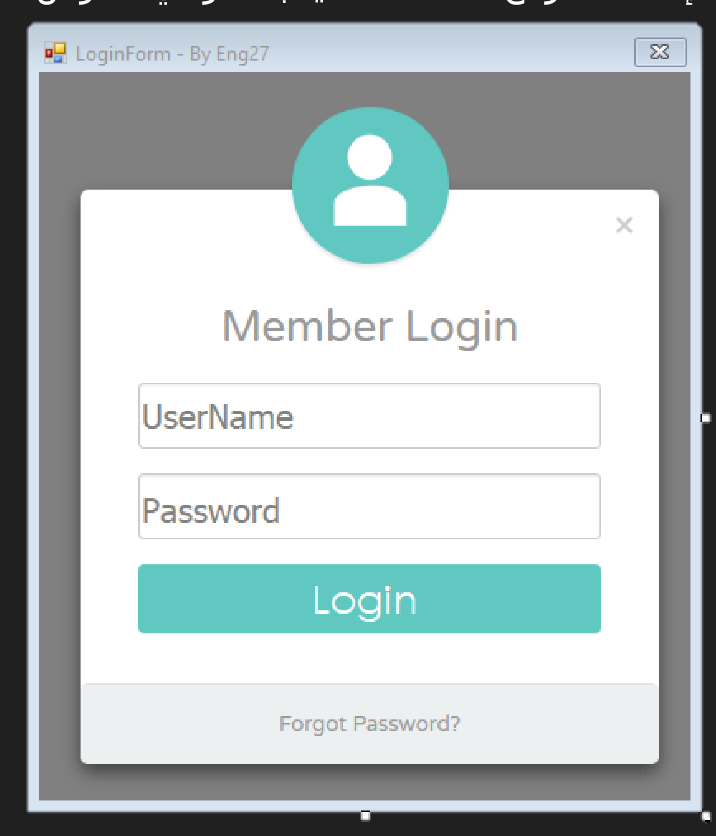
مزوّد الأخطاء ErrorProvider

تعطيك هذه الأداة إمكانية جميلة في إدارة أخطاء برنامجك وتقديمها للمستخدم، لا تتعلق بالخطأ البرمجي بحد ذاته وكيفية إزالته أو التعامل معه، وإنما تعنى بتقديم وصف للمستخدم عن الخطأ في تنفيذ أو فهم البرنامج الذي وقع به المستخدم عند أدوات معينة، كما يظهر إشارة بجانب الأداة على النموذج.

بمعنى أن هذه الأداة لا تقي البرنامج من الأخطاء البرمجية مثل try catch، وإنما تدير الأخطاء التي يقع بها المستخدم في استخدامه للبرنامج. وبالتالي فهذه الأداة الهدف منها إخبار المستخدم أنه لا يستخدم البرنامج بالشكل الصحيح، وتعطيه لمحة عن خطأه وكيفية إصلاحه من خلال رسالة تلميح بجانب الأداة التي أخطأ المستخدم فيها. أي أنها موجه للمستخدمين وليس المبرمجين.

بفرض أن لديك نافذة تسجيل دخول، بالشكل التالي:

طبعا ما تراه أمامك هو مجرد صورة وضعتُ فوقها الأدوات بعد إخفاء حدودها، وهو أسلوب يمكنك اعتماده لتصميم النوافذ. بإمكانك إنشاء نموذج Form عادي بصندوقي نصوص، وزر واحد. أضف الأداة ErrorProvider إلى مشروعك واستخدم الكود التالي:





```
using System.Windows.Forms;
using System;
using System.Drawing;

namespace WindowsFormsApplication
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); }

        private void button1_Click(object sender, EventArgs e)
        {
            // في البداية يجب مسح سجل الأخطاء السابقة
            errorProvider1.Clear();

            // في حال عدم وجود أخطاء يمكن تسجيل الدخول، هذا المتغير سيخبرنا بذلك
            bool LoginAccepted = true;

            // هنا سندير الأخطاء في البرنامج
            // أول خطأ محتمل هو أن يترك اسم المستخدم خالي
            // ثاني خطأ قد يقع به المستخدم هو أن يترك كلمة السر خالية
            // ثالث خطأ هو أن تكون كلمة السر أقل من 8 حروف (مثلًا)
            // يكون اسم المستخدم أو كلمة السر خاليين إذا كان لونهما رماديا
            // (يوجد في نهاية الكود بعض الإجراءات التي ستضبط اللون)
            // إذا حدث خطأ ما فإنه سيتم وضع إشارة بجانب الأداة التي حدث عندها الخطأ مع رسالة للمستخدم حتى يذوق على دمه
            if (textBox1.ForeColor == Color.Gray)
            {
                errorProvider1.SetError(textBox1, "You have to enter username");
                LoginAccepted = false;
            }

            if (textBox2.ForeColor == Color.Gray)
            {
                errorProvider1.SetError(textBox2, "You have to enter password");
                LoginAccepted = false;
            }

            if (textBox2.Text.Length < 8)
            {
                errorProvider1.SetError(textBox2,
                     "Password must be more than 8 characters");
                LoginAccepted = false;
            }

            // إذا لم يكن هناك أي خطأ فإن المتغير المنطقي سيخبرنا بذلك
            if (LoginAccepted)
                ShowMainForm();
        }
        void ShowMainForm()
        {
            // إظهار رسالة بتمام عملية التسجيل
            MessageBox.Show("Loging Accepted!!");
        }
```

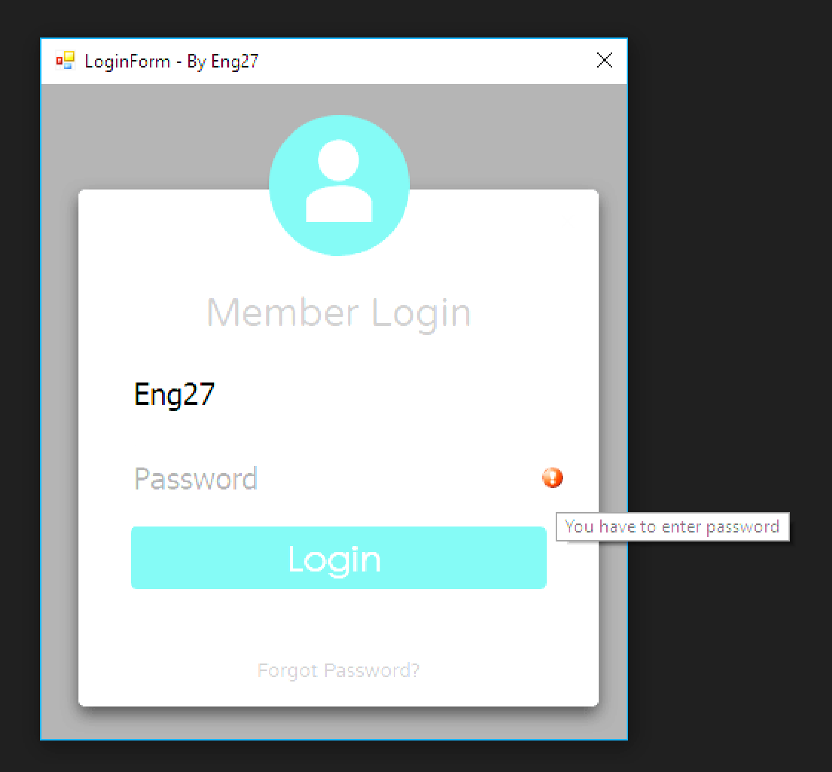
```
        // الإجراءات التالية ستقوم بتهيئة ألوان خطوط صناديق النصوص
        // بحيث يكون الخط رمادي إذا لم يكن صندوق النص فارغًا أو لم يكن على قيمته الافتراضية
        private void textBox1_Leave(object sender, EventArgs e)
        {
            // إذا كان اسم المستخدم خاليًا أو بقيمته الافتراضية "اسم المستخدم" يكون لونه رماديًا
            if (textBox1.Text == "UserName" || textBox1.Text == "")
            {
                textBox1.Text = "UserName";
                textBox1.ForeColor = Color.Gray;
            }
            else // وإلا، يكون أسودًا
                textBox1.ForeColor = Color.Black;
        }

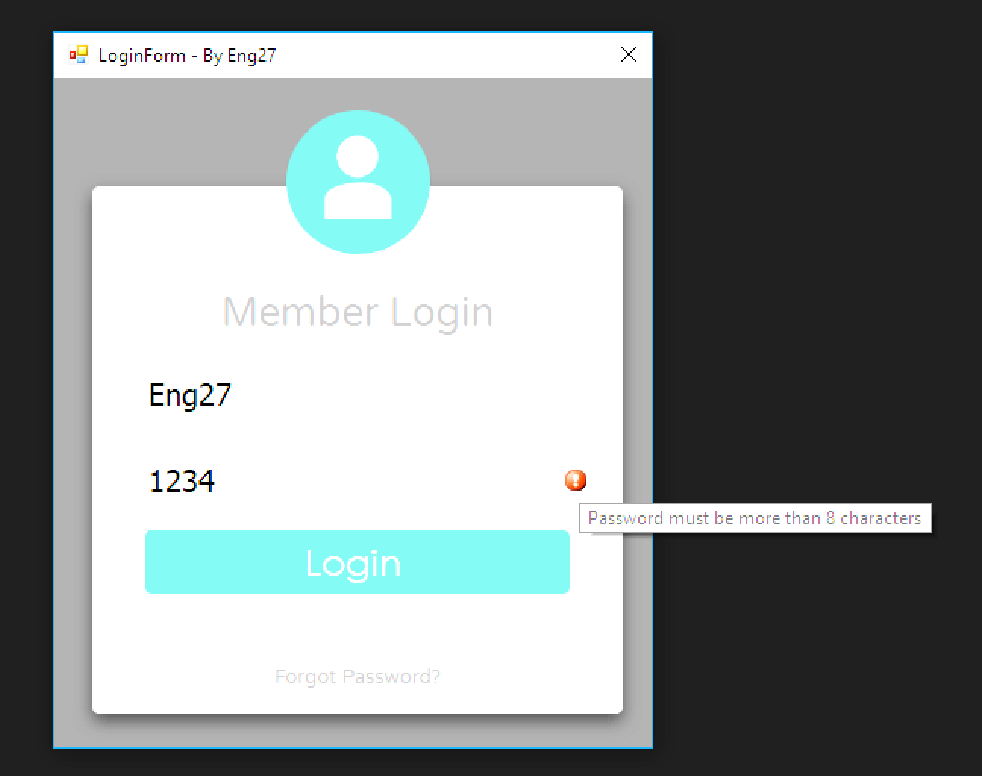
        private void textBox2_Leave(object sender, EventArgs e)
        {
            // بنفس طريقة الأداة السابقة يكون رماديًا
            if (textBox2.Text == "Password" || textBox2.Text == "")
            {
                textBox2.Text = "Password";
                textBox2.ForeColor = Color.Gray;
            }
            else // أو أسودًا
                textBox2.ForeColor = Color.Black;
        }

        // الإجراءات التالية يفرغان صناديق النصوص من محتواها عند نقل التركيز إليها
        private void textBox1_Enter(object sender, EventArgs e)
        {
            textBox1.Text = "";
            textBox1.ForeColor = Color.Black;
        }

        private void textBox2_Enter(object sender, EventArgs e)
        {
            textBox2.Text = "";
            textBox1.ForeColor = Color.Black;
        }
    }
}
```

جرب عدم إدخال اسم المستخدم أو كلمة السر أو كلاهما، أو أدخل كلمة سر أقل من ثمانية حروف، وسترى أن البرنامج لا يسمح لك بتسجيل الدخول، لا وفوقها يعطيك مكان الخطأ.





غايتي من هذا المثال ليس فقط الأداة التي نناقشها وحسب، وإنما التمهيد لأساليب تصميم الأدوات وأفكارها، فكما لاحظت فإننا استخدمنا صورة لم نصمم منها شيء على

أنها واجهة البرنامج، وجعلنا لصناديق النصوص قيمة افتراضية تظهر بلون رمادي تُعرف بالعلامة المائية Watermark.

مقياس الأداء PerformanceCounter

يمكنك من خلال هذه الأداة معرفة تفاصيل كثيرة عن أجهزة حاسوبك، مما يعطيك إمكانيات كثيرة لجمع المعلومات في حواسيب المستخدمين والاستفادة منها للحصول على أكبر فائدة ممكنة من البرنامج.

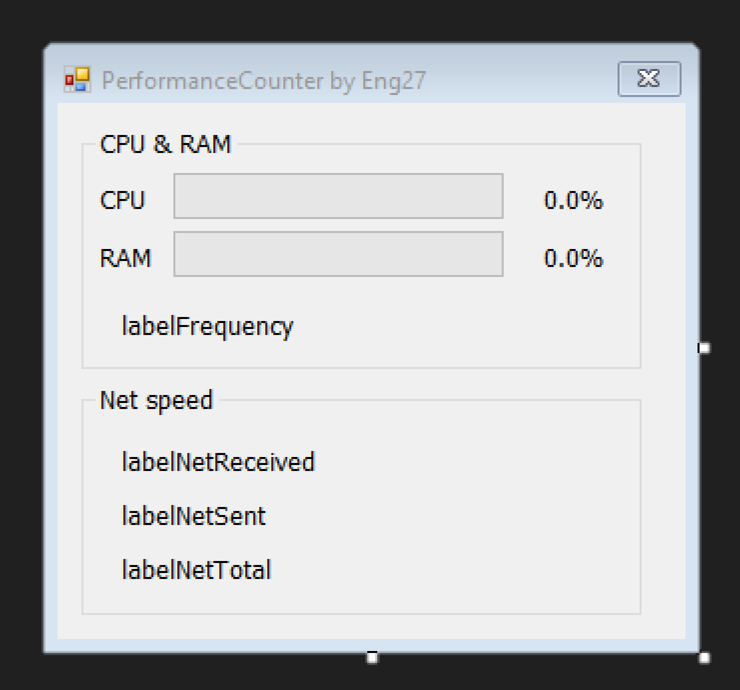
pcRAM System.Diagnostics.PerformanceCounter

Property	Value
(ApplicationSettings)	
(Name)	pcRAM
CategoryName	Memory
CounterName	% Committed Bytes In Use
GenerateMember	True
InstanceLifetime	Global
InstanceName	
MachineName	.
Modifiers	Private
ReadOnly	True

يتم ضبط الخاصية CategoryName على الجزء المراد قياسه ضمن أجهزة الحاسوب، وهي قيمة نصية String، أي أن ضبطها برمجيًّا يتطلب حفظ القيمة المراد ضبط الخاصية عليها بالكامل. ثم تُضبَط الخاصية CounterName على ما يجب قياسه ضمن الجزء المُقاس، وهي أيضًا قيمة نصية String.

في حال كان هناك أقسام أو نسخ من الجهاز المقاس يتم تحديده ضمن الخاصية InstanceName، مثل حالة التعامل مع الأقراص الصلبة، أو التعامل مع المعالج، أو غيرها.

وكمثال، صمم النافذة المجاورة:



أضف الأدوات التالية بالأسماء أدناه:

Form1	System.Windows.Forms.Form
groupBox1	System.Windows.Forms.GroupBox
groupBox2	System.Windows.Forms.GroupBox
label1	System.Windows.Forms.Label
label2	System.Windows.Forms.Label
labelCPU	System.Windows.Forms.Label
labelFrequency	System.Windows.Forms.Label
labelNetReceived	System.Windows.Forms.Label
labelNetSent	System.Windows.Forms.Label
labelNetTotal	System.Windows.Forms.Label
labelRAM	System.Windows.Forms.Label
pcCPU	System.Diagnostics.PerformanceCounter
pcFrequency	System.Diagnostics.PerformanceCounter
pcNetReceived	System.Diagnostics.PerformanceCounter
pcNetSent	System.Diagnostics.PerformanceCounter
pcNetTotal	System.Diagnostics.PerformanceCounter
pcRAM	System.Diagnostics.PerformanceCounter
progressBarCPU	System.Windows.Forms.ProgressBar
progressBarRAM	System.Windows.Forms.ProgressBar
timer1	System.Windows.Forms.Timer

غيّر خصائص مقاييس الأداء CPU وRAM وFrequency إلى ما يلي:

(Name)	**pcCPU**
CategoryName	**Processor**
CounterName	**% Processor Time**
GenerateMember	True
InstanceLifetime	Global
InstanceName	**_Total**

(Name)	**pcRAM**
CategoryName	**Memory**
CounterName	**% Committed Bytes In Use**
GenerateMember	True
InstanceLifetime	Global
InstanceName	

(Name)	**pcFrequency**
CategoryName	**Processor Information**
CounterName	**Processor Frequency**
GenerateMember	True
InstanceLifetime	Global
InstanceName	**_Total**

غيّر قيمة خاصية Interval المؤقت لثانية واحدة، واتركه غير مفعَّل، ثم استخدم الكود التالي:



```
using System;
using System.Diagnostics;
using System.IO;
using System.Windows.Forms;

namespace PerformanceCounter
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            string WifiName = GetWifiName();

            pcNetReceived.CategoryName = "Network Adapter";
            pcNetSent.CategoryName = "Network Adapter";
            pcNetTotal.CategoryName = "Network Adapter";
            pcNetReceived.CounterName = "Bytes Received/sec";
            pcNetSent.CounterName = "Bytes Sent/sec";
            pcNetTotal.CounterName = "Bytes Total/sec";
            pcNetReceived.InstanceName = WifiName;
            pcNetSent.InstanceName = WifiName;
            pcNetTotal.InstanceName = WifiName;

            timer1.Enabled = true;
        }

        string GetWifiName()
        {
            var f = File.Create(Application.StartupPath + "\\SI.bat");
            f.Close();
            StreamWriter FileW =
                 File.AppendText(Application.StartupPath + "\\SI.bat");
            FileW.WriteLine("systeminfo > SI.dat");
            FileW.Close();
            Process p = new Process();
            ProcessStartInfo pInfo = new ProcessStartInfo(f.Name);
            p.StartInfo = pInfo;
            p.Start();
            p.WaitForExit();
            p.Close();

            string[] SysInfo =
                 File.ReadAllLines(Application.StartupPath + "\\SI.dat");
            string wifi = SysInfo[37].Split(':')[1];
            wifi = wifi.Remove(0, 1);
            if (string.IsNullOrWhiteSpace(wifi))
                return null;
            else
                return wifi;
        }
```

```
        private void timer1_Tick(object sender, EventArgs e)
        {
            float fcpu = pcCPU.NextValue();
            float fram = pcRAM.NextValue();
            progressBarCPU.Value = (int)fcpu;
            progressBarRAM.Value = (int)fram;
            labelCPU.Text = string.Format("{0:0.0}%", fcpu);
            labelRAM.Text = string.Format("{0:0.0}%", fram);

            float fReceived = pcNetReceived.NextValue() / 1024;
            float fSent = pcNetSent.NextValue() / 1024;
            float fTotal = pcNetTotal.NextValue() / 1024;
            labelNetReceived.Text =
                string.Format("Received: {0:0.0} KB/s", fReceived);
            labelNetSent.Text = string.Format("Sent: {0:0.0} KB/s", fSent);
            labelNetTotal.Text = string.Format("Total: {0:0.0} KB/s", fTotal);

            float fFrequency = pcFrequency.NextValue();
            labelFrequency.Text =
                string.Format("Processor frequency: {0:0.0} Hz", fFrequency);
        }
    }
}
```

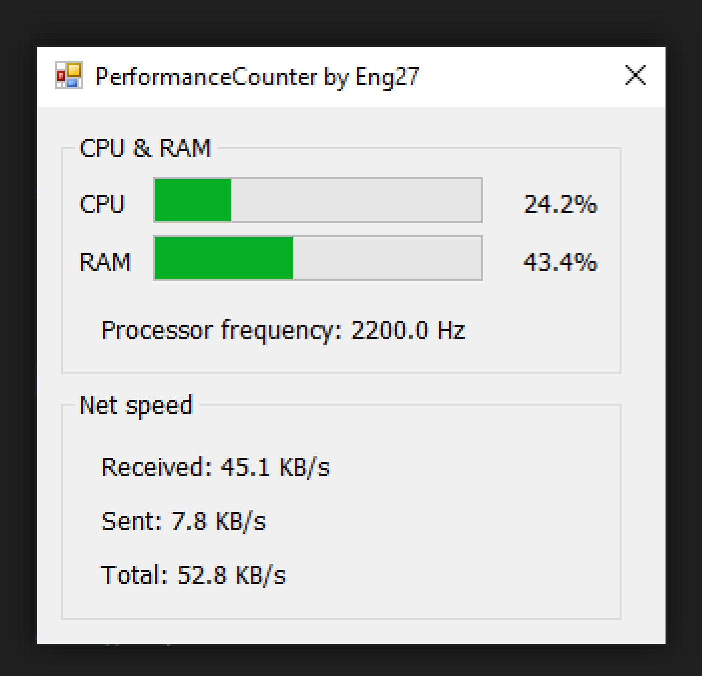
يقوم التابع GetWifiName بإيجاد اسم كرت الشبكة Network Card الرئيسي وذلك اعتماداً على أوامر موجّه الأوامر، والتي بيّنا كيفية استخدامها في الكتاب الأول. يتم توليد ملف من نوع dat يحوي ناتج تنفيذ الأمر systeminfo، ويأخذ السطر رقم 38 وهو اسم كرت الشبكة الرئيسي في الحاسوب.

بتشغيل البرنامج سيحدث مايلي: سيعمل cmd بأمر systeminfo:

```
Select C:\Windows\system32\cmd.exe

E:\C#\tests\PerformanceCounter\PerformanceCounter\bin\Debug>systeminfo  1>SI.mpd
Loading Processor Information ...
```

ثم ستعمل نافذة برنامجك:



إنشاء الأدوات برمجيًّا

يمكنك إنشاء الأدوات من خلال سحبها وإفلاتها على النافذة أثناء تصميمها، كما يمكنك ذلك من خلال استخدام بعض الأكواد البرمجية. ستجد هذا الأسلوب نافعًا عند حاجتك لإنشاء أدوات برمجية أثناء التنفيذ Run-Time، كما أنه سيعني لنا الكثير على امتداد هذا الكتاب، خصوصًا كأساسيات ومفاهيم لمنطلقات بحثنا.

إنشاء الأدوات من خلال الكود هو عبارة عن استنساخ كائنات من فئات معينة، وأعتقد أنه يدور في بالك أن هذه الفئات هي هذه الأدوات لكن مكتوبة بصيغة عامة، أعني أن هذه الفئات هي قالب عام للأدوات التي ستمثلها، والكائنات ماهي إلا تخصيص لهذه القوالب. وعلى اعتبار أنها استنساخ لكائنات، فهذا يعني أنه لإنشاء أداة ما يجب استخدام كود يشابه هذا:

```
Control controlName = new Control();
```

حيث أن Control هي اسم الفئة التي سيتم استنساخها لإنشاء الأداة، أي أنها تمثل الأداة كمفهوم، وcontrolName الاسم البرمجي للأداة.

لاحظ أنه بإنشاءك للأداة لن تظهر مباشرة على النموذج Form (أو يمكننا أن نسميها النافذة)، فهي مثلها مثل غيرها من المتغيرات مكانها في الذاكرة ليس إلا، ولإظهارها على النافذة عليك إضافتها أولًا، بمعنى أنك تنشئ الأداة في الذاكرة ثم تضيفها للنافذة.

فمثلًا لإنشاء وإضافة زر Button لنافذة برنامجك استخدم الكود التالي:



```
private void button1_Click(object sender, EventArgs e)
{
    Button b = new Button();
    this.Controls.Add(b);
}
```

وبتنفيذ الكود ستحصل على زر موجود في الزاوية اليسرى العلوية، تحديدًا في الموقع Location: (0, 0) من النافذة.



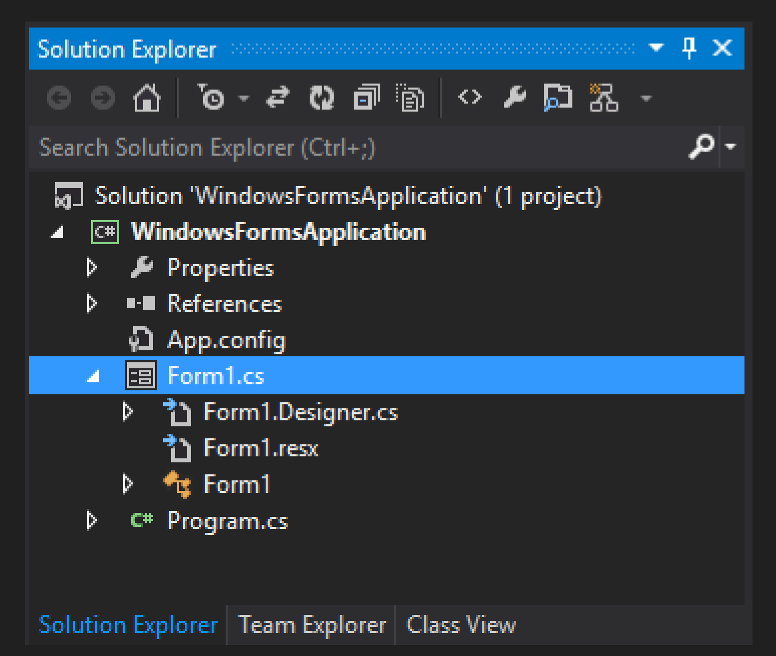
في الكود السابق كان الاسم البرمجي للزر المُضاف name = b، في حين أن بقية الخصائص لها القيم الافتراضية (فالنص الظاهر Text لا يحمل قيمة والموقع (0, 0) والحجم الافتراضي وغيرها من الخصائص).

وقد أفاجِئُك إذا أخبرتك أنه يمكنك إنشاء نموذج Form كامل من خلال الأكواد، قد يُخيَّل إليك أن الموضوع ببساطة إنشاء زر من خلال كود مكون من سطرين، ولكن الموضوع أعقد من ذلك، فأنا لا أتحدث عن إنشاء نموذج ثانوي فقط، أنا أتحدث عن إنشاء نموذج دون وجود أية نماذج أخرى في البرنامج، وللمزيد أحيلك لكتاب "فيجوال ستوديو 2008" لـ أحمد جمال خليفة، الصفحة 328.

للمزيد راجع [هذا](https://docs.microsoft.com/en-gb/dotnet/framework/winforms/how-to-create-a-windows-forms-application-from-the-command-line) [1] الرابط.

[1] راجع مقال لمايكروسوفت عن إنشاء نافذة من خلال الكود:
https://docs.microsoft.com/en-gb/dotnet/framework/winforms/how-to-create-a-windows-forms-application-from-the-command-line

مصمم النموذج designer



كبداية أضف أداةً ما لنموذج برنامجك ولتكن زر أوامر Button، وانتقل مباشرة لمتصفح المشروع Solution Explorer:

وسّع ملفات النموذج Form1 لتحصل على ثلاثة ملفات، يهمنا منها الملفين Form.Designer.cs وForm، انقر على كل ملف منهما مرتين لفتحهما، لتجد أن Form هو نفسه النموذج المعروف وأكواده هي أكواد برنامجك، بينما Form.Designer هو شيء آخر تمامًا ومن المحتمل هو أنك تستكشفه للمرة الأولى..

كود النموذج:

```
using System;
using System.Windows.Forms;

namespace WindowsFormsApplication
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
    }
}
```

كود المصمم:



```
namespace WindowsFormsApplication
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
        /// otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #Windows Form Designer generated code#

        private System.Windows.Forms.Button button1;
    }
}
```

تعال نناقش الكودَين كلًّا على حدة.. كود النموذج ببساطة يقوم باعتماد مكتبتين كمرجع له، بالبداية يهيئ مكونات البرنامج، وعند النقر على الزر button1 يُغلق البرنامج. أما كود المصمم designer ففي بدايته تم تعريف متغير باسم components مساوٍ للقيمة null، والذي يؤدي باختصار إلى حذف المكونات عند الخروج من النموذج حتى لا تبقى في الذاكرة، ثم يوجد إجراء يقوم بتنظيف الذاكرة، ثم منطقة Region فيها كود بعنوان: Windows Form Designer generated code، ثم كود إنشاء زر برمجيًّا كما فعلنا في الفقرة السابقة.

والآن بعد هذه المناقشة، أرغب بطرح بعض الأسئلة عليك، ولعل بعضها قد انبثق بجانب رأسك:

- ما الغاية من الإجراء InitializeComponent في الكود الأول؟
- ما هو معنى الملاحظات الكثيرة في الكود الثاني؟ ولماذا تبدأ بثلاثة إشارات /// بدلًا من اثنتين //؟؟
- ماذا يوجد في المنطقة Windows Form Designer generated code؟؟؟
- كيف تمت إضافة كود إنشاء button1؟؟؟؟

سأناقش هذه الأسئلة بفقرات فرعية، لتنظيم الأفكار.

الإجراء InitializeComponent

من المرجّح أنه وأثناءَ تعلمك لأساسيات التعامل مع النوافذ في C# قد صادفت هذا الإجراء بين أسطر الشروحات وفي مشاهد الفيديوهات التي تشرح اللغة، وغالبًا ما كان شرح هذا التابع بأنه "تابع هام جدًا، إياك ولمسه! يمنع الاقتراب والتصوير"، أو شيء مثل "قد يثير اهتمامك الإجراء InitializeComponent، لا عليك صادقه في برامجك وأعطه بعض الاحترام، هو إجراء يقوم بتهيئة أدوات مشروعك عند بداية تنفيذ البرنامج". فما وظيفة هذا الإجراء بالتحديد؟؟ وهل فعلًا لا يمكن الاستغناء عنه لهذه الدرجة؟

الجواب هو نعم، هو إجراء هام وما قيل لك صحيح. يقوم هذا الإجراء بتهيئة مكونات النموذج لتأخذ شكلها الذي صممت عليه، فالمكونات كما قلنا قبل صفحات يتم إنشاؤها أولًا ثم إضافتها، وكما رأينا فإن هذه المكونات تُضاف بخصائصها الافتراضية. هذا يعني أن هذا الإجراء هو المسؤول عن إعطاء مكونات البرنامج شكلها ومواصفاتها، لذلك لا يمكن الاستغناء عنه.

أما ما يقوم به بالتحديد هذا السطر البرمجي فهو ببساطة – وكما تلاحظ – يستدعي إجراءً باسم InitializeComponent والأخير يقوم بواجبه. طيب، أين هذا الإجراء؟ إذا كان تفكيرك بوليسيًّا فربما المنطقة Windows Form Designer generated code كانت أول المتهمين بالنسبة لك، والتي سنناقشها في الفقرة التالية.

المنطقة Windows Form Designer generated code

كما تعلم فإنه يمكن استخدام ما يسمى بالمناطق regions لتنظيم الكود، وعندها يتم عنونة الكود باسم معين، وإخفاؤه خلف هذا الاسم، بحيث يُعرض الكود عند توسيع الاسم ويُخفى عند تضمينه. وبعض الفصول القادمة سترى في أكوادها مناطق كثيرة لتنظيم الكود، كالفصول السادس والسابع والثامن.

في كود المصمم ستلاحظ هذه المنطقة، عنوان محاط بمستطيل، وبجانبه إشارة + لتوسيعه.. لاحظ أن رقم سطر هذه المنطقة 23، والسطر الذي يليه رقمه 60. هل تفكر فيما أفكر فيه؟ (لا أقصد السيطرة على العالم، أقصد أن هناك أسطر مخفية).

```
23 ⊞        Windows Form Designer generated code
60
```

اضغط على إشارة + لتوسيع الكود ولتحصل على:



```
#region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.ForeColor = System.Drawing.Color.Black;
            this.button1.Location = new System.Drawing.Point(81, 94);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(75, 23);
            this.button1.TabIndex = 0;
            this.button1.Text = "button1";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackColor = System.Drawing.Color.White;
            this.ClientSize = new System.Drawing.Size(368, 305);
            this.Controls.Add(this.button1);
            this.ForeColor = System.Drawing.Color.White;
            this.Name = "Form1";
```

```
            this.Text = "Form1";
            this.ResumeLayout(false);
        }
        #endregion
```

أعلم أنك الآن تنظر إلى ذاك الإجراء نظرة ارتياب واستغراب واستفهام، نعم نعم أنا أقصد InitializeComponent، إنه هو عزيزي القارئ!!

تأمله وتفحصه ودقق في سطوره، الظاهر أنه يعيد استنساخ button1، ثم يستدعي تابعًا ما، ثم يقوم بتغيير بعض خصائص الكائن button1، كما أنه يربط حدث button1_Click بحدث النقر على الزر button1 أي button1.Click()، ثم يغير بعض خصائص الفورم..

هل لاحظت كيف وُجدت الأكواد التي تسببت بظهور كل من النافذة والزر على شكلهما؟ (جرب غير بعض خصائص النافذة أو الزر، غير الاسم أو اللون أو غيرها، وعد إلى هذا الكود، ولاحظ ما سيحدث). وبرأيي أنه باتت لديك فكرة جيدة عن وظيفة هذا الإجراء، بل من المحتمل أنك لم تعد تتخيل كيف يمكن للبرنامج أن يضيف الأدوات إليه ويهيئها لتظهر بالشكل الذي تم تصميمها عليه لولا وجود هذا الإجراء!!

لاحظ مقدار التنظيم والترتيب في هذا التابع، وتأمل ما سيحدث لو أننا أضفنا عشرات الأدوات وغيرنا الكثير من خصائصها!!

الآن باتت لديك صورة واضحة عن كيفية تعامل #C مع الأدوات، كل ما تراه أمامك هو مجرد أكواد ليس إلا، حتى الأدوات مهما كانت معقدة بشكلها ومضمونها ومحتواها كـ DataGridView مثلًا هي كود ليس إلا!! عيب عليك من الآن فصاعدًا أن تعتقد أن هناك عملية سحرية تقوم بها #C لربط البرنامج بالأدوات 😒.

طيب ماذا بشأن الملاحظة في بداية الكود: "طريقة مطلوبة لدعم التصميم – لا تعدل محتوى هذه الطريقة باستخدام محرر الأكواد"؟ لا بد أن هذه الملاحظة – الأشبه بالتنبيه – قد حركت فيك جيناتك التي كانت تحثك على دق الأبواب والهروب عندما كنت في الابتدائية 👦🏻🐒، فجال بخاطرك أنه لماذا لا يمكن تعديل المحتوى باستخدام محرر الأكواد (الفيجوال ستوديو مثلًا) 🤔؟

على كل حال فهذه الملاحظة غير موجهة لمن يقرأ هذه الفقرة، وأنت محشوم عزيزي القارئ فقد قرأتها لتوّك، ومعناها أن هذا الإجراء مهم لعملية التصميم (ترتيب الأدوات وتعديل خصائصها) وهذا ما ناقشناه في هذه الفقرة، أما غايتهم من ضمان عدم تعديل المحتوى باستخدام محرر الأكواد فهو ببساطة حتى لا يحدث أخطاء، أما أنت فلك باع طويل مع InitializeComponent لذلك فلا يتوقع حدوث أخطاء منك (لكن مع هذا فأحذرك، بعض الأخطاء قد تدمر المشروع ولا يمكن التراجع عنها، لا تخبص [1] وتلزقها برقبتي في النهاية)، ما أحاول قوله: يمكنك تعديل محتوى هذا الإجراء لكن بحدود وقيود ولحكمة.

على أي أساس تُضاف أكواد إنشاء الأدوات في المصمم؟

كما لاحظت في كود المصمم فإنه قد تم إضافة كود إنشاء الكائن button1 تلقائيًّا، وهذا يعني أنه لو كانت هناك أدوات أخرى تنشؤها فسيتم إضافة أكواد استنساخها في نهاية كود المصمم، تمامًا كـ button1. وأعتقد أن جواب هذه الفقرة لديك أساسًا خصوصًا بعد قراءة الفقرتين السابقتين، ومع ذلك فإنه من المفيد مناقشة بعض النقاط في فقرة مخصصة لكود إنشاء الأدوات.

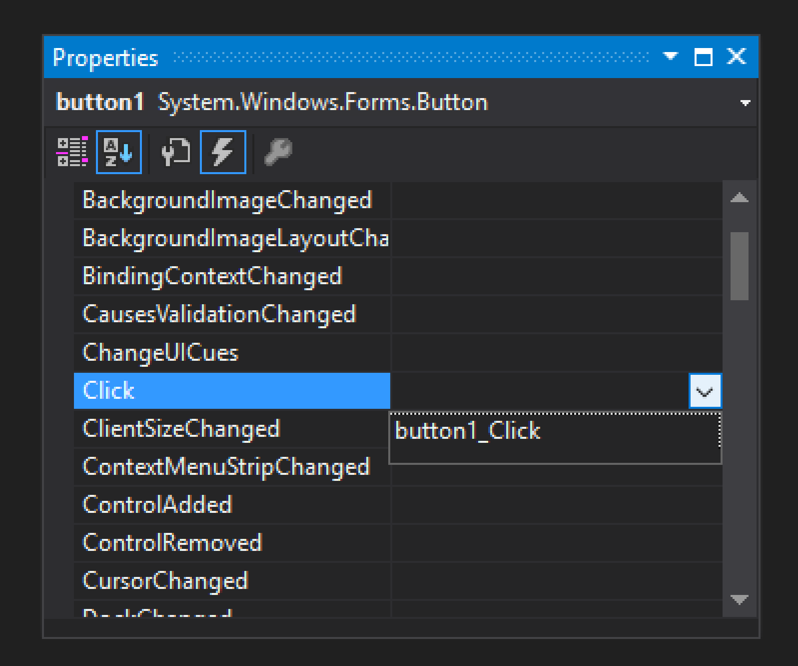
بالإضافة إلى أنه سيتم إضافة كود – في نهاية كود المصمم – لإنشاء كل أداة استخدمتها في تصميم نموذجك، فإنه يتم تفصيل هذه الأدوات في الطريقة InitializeComponent، وفيها يتم ربط أحداث الأداة بالأداة، وهذه نقطة مهمة جدًّا، وبفهمك إياها يمكنك السيطرة على برنامجك – في طور البرمجة والتصميم – بشكل أفضل.

عند حذف أحد أحداث الأدوات فإن فيجوال ستوديو سيرسل لك خطأً يعكر صفو يومك، وهذا بسبب أنك قد حذفت الحدث دون حذف جذره من المصمم، لذلك فمن الآن وصاعدًا يمكنك وبكل ثقة حذف روابط الأحداث في المصمم ثم حذفها من الكود دون حدوث خطأ واحد حتى.. جرب احذف الحدث button1_Click من كود النموذج ليفاجئك الفيجوال ستوديو بخطأ يحمل في طياته كميات كبيرة من النكد والشؤم، قم بالتراجع عن فعلتك واستعد الحدث، ثم انتقل لكود المصمم واحذف فيه الكود الذي تم فيه ربط button1.Click

[1] لا تخبص: لا تتصرف بطريقة غير مدروسة.

بالحدث button1_Click والذي ستجده غالبًا في نهاية الجزء من الكود الذي يوصف الأداة، ثم اذهب لحذفه دون أن يجرؤ الفيجوال ستوديو على التحدث بكلمة واحدة!

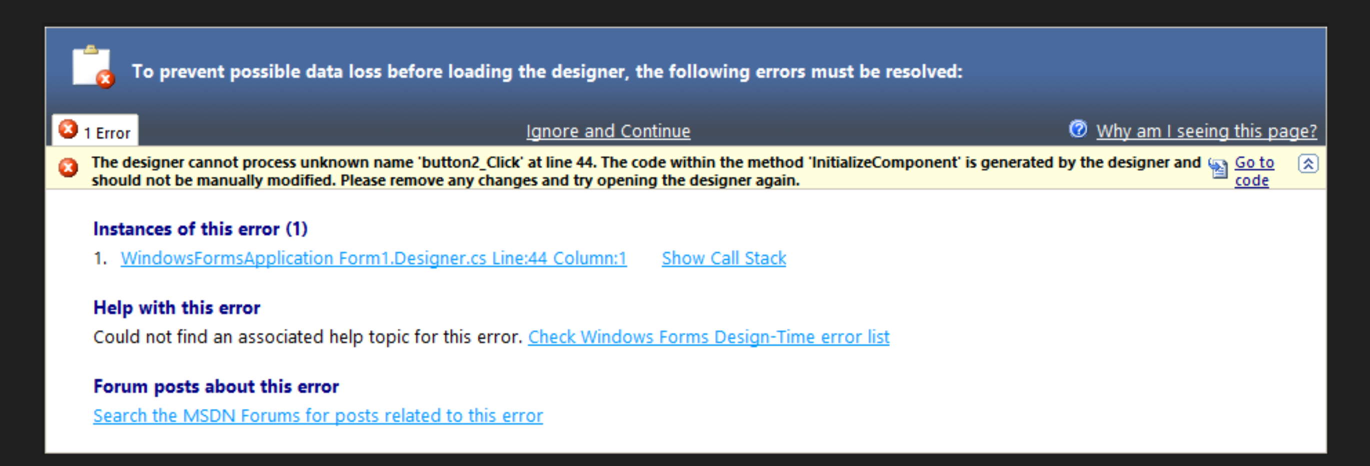
وعلى سيرة أخطاء فقدان الإجراءات المربوطة مع الأدوات، فقد تعاني كثيرًا عند نسخ الأكواد من الإنترنت ولصقها في برنامجك لتجد أنها لا تعمل، أو كما حصل مع كثيرين من قراء كتابي السابق في المشاريع الجاهزة عندما نسخوا أكوادها إلى مشاريعهم، دون ربط الأحداث بالأدوات. ولحل هذه المعضلة يمكنك تعديل كود المصمم وربط الحدث بالأداة (هذا ما عنيته بإمكانية تعديل كود المصمم عند وجود حكمة من ذلك)، أو ببساطة إضافة حدث للأداة في قسم أحداث وخصائص الأدوات، بالشكل التالي:



وأما تعديل كود المصمم فهو كما يلي (بفرض أننا نرغب بربط حدث النقر على الأداة بحدث النقر على أداة ثانية، ولتكن button2، بفرض وجودها):

```
this.button1.Text = "button1";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button2_Click);
//
// button2
```

وبخصوص الخطأ الذي يحصل عند التلاعب بالمصمم – في حال لم يتدمر المشروع وتفقد إمكانية التراجع – أو الذي يحصل عند نسخ الأكواد نسخًا أعمى (مجرد نسخ ولصق دون فهم ما يحدث) فهو هذا:



وحله وببساطة ورواق النقر على سطر الخطأ داخل الفقرة Instances of this error لينقلك مباشرةً لمكان الخطأ:

```
//
// button1
//
this.button1.ForeColor = System.Drawing.Color.Black;
this.button1.Location = new System.Drawing.Point(81, 94);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(75, 23);
this.button1.TabIndex = 0;
this.button1.Text = "button1";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button2_Click);
//
// button2
//
this.button2.Location = new System.Drawing.Point(0, 0);
this.button2.Name = "button2";
this.button2.Size = new System.Drawing.Size(75, 23);
this.button2.TabIndex = 1;
this.button2.Text = "button2";
this.button2.UseVisualStyleBackColor = true;
this.button2.Click += new System.EventHandler(this.button2_Click);
```

وكما هو واضح فلدينا خطأين، وكلاهما بسبب فقدان الحدث button2_Click، والحل ببساطة حذف الأسطر التي تربط الأدوات بهذا الحدث، أو الانتقال لكود النافذة وإنشاء هذا الحدث، أو استعادته في حال فقدانه بطريقة أو بأخرى.

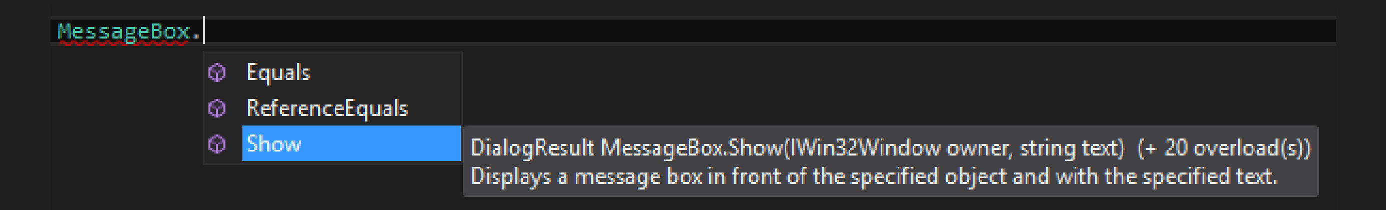
لاحظ أن كود المصمم يتم تحديثه تلقائيًا مع كل حركة تخطوها في نافذة برنامجك. فمثلًا لو كان عندي زر باسم button2 لوُجد له جزءًا من الكود يوصّفه كما في الصورة الأخيرة، ولو حذفت الزر من النموذج وعدت للمصمم لوجدت أن الجزء الذي كان يوصّف الزر قد اختفى!!

هذه الديناميكية في تصميم الأدوات هي بفضل الطريقة SuspendLayout، الموجودة في الأسطر الأولى للإجراء InitializeComponents، والتي تجبر الفيجوال ستوديو على تمثيل أكواد المصمم على النافذة وأدواتها في طور التصميم Design-Time. كما أن الطريقة ResumeLayout تُعلِم الفيجوال استوديو أن عملية التصميم قد انتهت، وذلك عند تمرير القيمة false إليها.

التوثيق XML Documentation

يُقصد بالتوثيق شرح ووصف أجزاء البرنامج – ككود – وذلك باستخدام التعليقات، وتُضاف باستخدام إشارة التعليق العادية لكن ثلاث مرات بدلًا من مرتين في بداية السطر، أي باستخدام /// في بداية السطر المراد إضافة التوثيق له.

عمومًا أنت تقابل توثيقات كثيرة، وتتعامل معها وتستفيد منها دون أن تدري!! تابع معي.. اكتب MessageBox ثم نقطة ليظهر لك ما يمكنك القيام به باستخدام هذه الفئة، انتقل إلى التابع Show لتحصل على التلميح التالي:



السطر الثاني يشرح لك ما يمكن للتابع أن يقوم به، وهو عبارة عن توثيق لهذا التابع!!

الآن اختر هذا التابع وافتح قوس، لتحصل على:

```
MessageBox.Show(
▲ 1 of 21 ▼ System.Windows.Forms.DialogResult MessageBox.Show(string text)
Displays a message box with specified text.
text: The text to display in the message box.
```

لاحظ أن الشرح بات يتضمن تفاصيل معينة عن البارامتر [1] المحدد، وهذا أيضا عبارة عن توثيق!

بناءً على ما تقدم فإن التوثيق XML Documentation أصبح واضحًا بالنسبة لك، أين يوجد ولماذا يستخدم، ولكن ماذا عن كيفية إنشاءه؟؟

الموضوع بسيط، فكما أسلفنا في بداية هذه الفقرة فالتوثيق ما هو إلا تعليق برمجي مبدوء بـ /// عوضًا عن //، وفي الواقع هذا ليس الفارق الوحيد بين التوثيقات والتعليقات البرمجية، فالتوثيقات تضاف تلقائيّا حسب الشيء الذي توثقه، أما التعليقات فتضاف يدويّا ولا قيود عليها. كما أن التوثيقات تظهر عند إسناد مكتبة الأدوات التي تحوي على طرق وفئات موثّقة، بينما لن تظهر التعليقات إلا إذا تم استعراض الكود بشكل شخصي.

في البداية دعنا ننشئ فئة، وفيها بعض الطرق والخصائص، ونراها دون توثيق. أنشئ مشروعًا جديدًا من نوع Class Library (يمكن إنشاء فئة class داخل المشروع نفسه وسنحصل على النتيجة ذاتها، لكن لتعميم الفكرة، ومناقشة بعض الأمور التي ستواجهك مستقبلًا مع التوثيقات في حال كانت ضمن ملفات dll منفصلة عن برنامجك، سننشئ مشروعًا جديدًا).

أنشئ مشروعًا باسم MyDLL مثلًا، فيه فئة تقوم بحل معادلات من الدرجة الثانية باسم Equation2. استخدم الكود التالي:

```
using System;

namespace MyDLL
{
    public class Equation2
    {
```

[1] البارامتر: الوسيط.

```
// المتغيرات التي ستلزمنا على امتداد عمل المكتبة
private double a, b, c; // ثوابت المعادلة
private double delta; // مميز المعادلة
private double x1, x2; // جذور المعادلة
private int round = 1; // عدد الأرقام بعد الفاصلة
private bool areParametersEntered; // متغير يحدد فيما إذا كانت جميع الثوابت مدخلة أو لا

public Equation2()
{
    areParametersEntered = false; // إذا استنسخنا الفئة دون وسطاء فهذا يعني أن الثوابت غير مُدْخَلَة
}

public Equation2(double A, double B, double C)
{
    // أما إذا استنسخناها مع وسطاء فالثوابت موجودة معناها
    a = A;
    b = B;
    c = C;
    areParametersEntered = true;
}

// حل المعادلة
public string Solve()
{
    if (areParametersEntered) // إذا كانت البارامترات كلها موجودة
    {
        // فيمكن الحل بتطبيق خوارزمية حل معادلة درجة ثانية بكل يسر وسهولة
        delta = b * b - 4 * a * c;
        if (delta >= 0)
        {
            x1 = (-b + Math.Sqrt(delta) / (2 * a));
            x2 = (-b - Math.Sqrt(delta) / (2 * a));
            return string.Format("X1 = {0}, X2 = {1}",
                Math.Round(x1,round),
                Math.Round(x2, round));
        }
        else
            return "Impossible equation";
    }
    else // وإن لم تكن البارامترات موجودة فيُعيد التابعة قيمة فارغة
        return string.Empty;
}

// هل المعادلة قابلة للحل أم لا
public bool isSolvable()
{
    if (areParametersEntered) // إذا كانت جميع البارامترات موجودة
        if (delta >= 0) // والمميز ليس سالبا
            return true; // فأهلا وسهلا
        else // وإن كان سالبا
            return false; // فمع السلامة
    else // أما إذا لم تكن البارامترات موجودة
        return false; // (بصوت مترجم اللغة الكوميالية) $#%$@$#
}
```

```
        // يحدد هذا الإجراء فيما إذا كانت جميع البارامترات موجودة
        void hasFullParameters()
        {
            // إذا كانت جميع الثوابت أرقام، بمعنى أنها ليست غير عددية
            if (!double.IsNaN(a) & !double.IsNaN(b) & !double.IsNaN(c))
                areParametersEntered = true; // هذا يعني أن جميع البارامترات موجودة
            else // وإلا
                areParametersEntered = false; // فغير موجودة
        }

        // خصائص الفئة:
        // في كل مرة يتم إدخال خاصية جديدة (والتي تمثل ثابت من ثوابت المعادلة، أو بارامتراتها) يجب التأكد من اكتمال المعطيات
        public double A
        {
            get { return a; }
            set { a = value; hasFullParameters(); }
        }

        public double B
        {
            get { return b; }
            set { b = value; hasFullParameters(); }
        }

        public double C
        {
            get { return c; }
            set { c = value; hasFullParameters(); }
        }

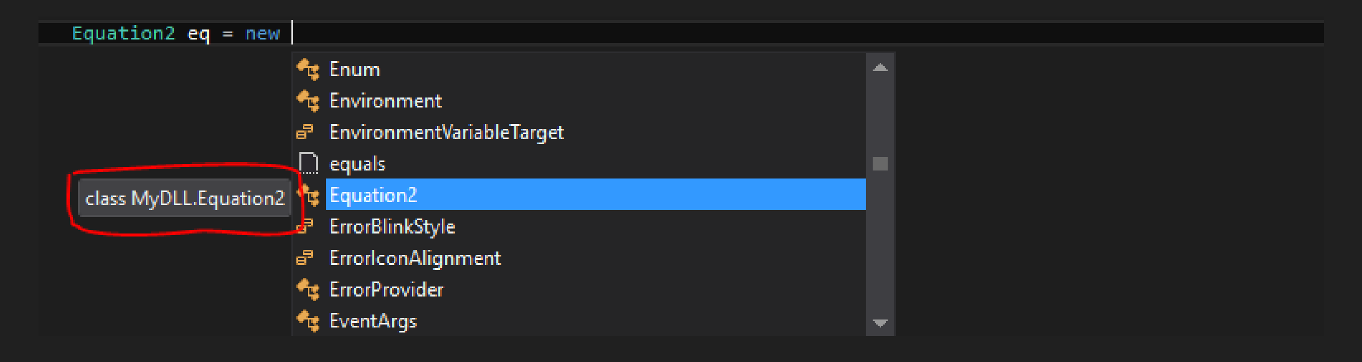
        public double Delta
        {
            get { return delta; } // لا يمكن إدخال المميز، لأنه لن يكون هناك طعمة للبرنامج بعدها
        }

        public int Round
        {
            get { return round; }
            set { round = value; }
        }
    }
}
```

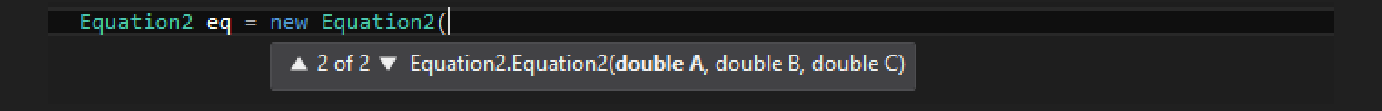
بغض النظر عن خوارزمية الكود، انقله كما هو فحسب، فالغاية ليست مناقشة النتائج وإنما أمر آخر. حتى لو لم تفهم الكود لا مشكلة، انسخه كما هو (يمكنك إذا أردت قراءة التعليقات المكتوبة بالكود بالتسلسل علّك تفهم ما يجري).

أنشئ مشروعًا من النوع WindowsFormsApplication واجعل MyDLL مرجعًا من مراجعه وصمم نافذة فيها زر واحد فقط، وتأمل معي:

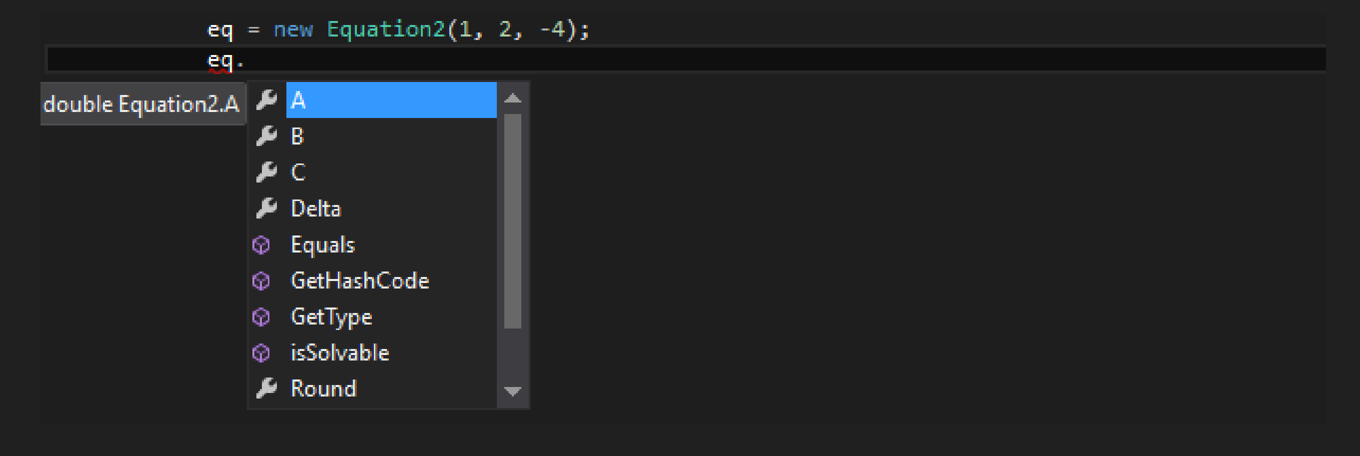
في البداية سننشئ كائنًا من النوع Equation2، لاحظ عدم وجود شرح أو تلميح عند ظهور الرسالة عند اختيار الفئة:



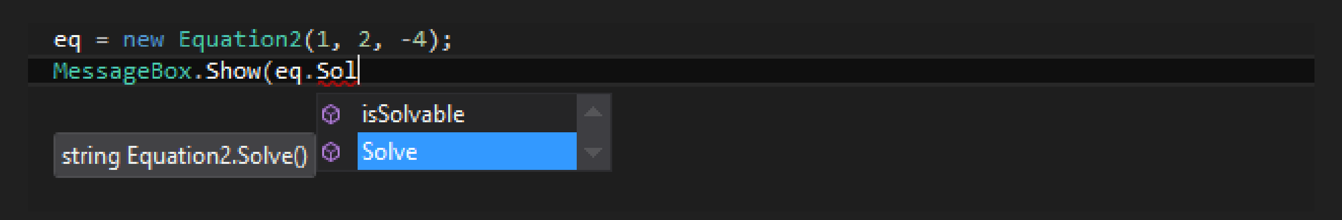
حتى عند إدخال وسطاء الفئة:



أو عند الوصول لخصائص الكائن:



أو عند استدعاء توابع الكائن:



بماذا ميكروسوفت أفضل منك حتى تضع تلميحات وتوثيقات لفئاتها وأنت لا تفعل؟؟

أضف بعض التوثيقات لكود MyDLL ليصبح بالشكل التالي:



لإضافة توثيق لمحتوى برمجي معين (فئة أو خاصية أو طريقة أو ...) انقل المؤشر إلى قبل هذا المحتوى بسطر واكتب /// ليتم إضافة توثيق خاص بهذا المحتوى (لكل محتوى توثيق خاص به).



```
using System;

namespace MyDLL
{
    /// <summary>
    /// Solves quadratic equations
    /// </summary>
    public class Equation2
    {
        private double a, b, c;
        private double delta;
        private double x1, x2;
        private int round = 1;
        private bool areParametersEntered;

        /// <summary>
        /// Creates an instance to Equation2 class.
        /// </summary>
        public Equation2()
        {
            areParametersEntered = false;
        }

        /// <summary>
        /// Creates an instance to Equation2 class with parameters.
        /// </summary>
        /// <param name="A">First constant.</param>
        /// <param name="B">Second constant.</param>
        /// <param name="C">Third constant.</param>
        public Equation2(double A, double B, double C)
        {
            b = B;
            c = C;
            if (a != 0)
            {
                a = A;
                areParametersEntered = true;
            }
        }

        /// <summary>
        /// Solves the equations and calcs its roots.
        /// </summary>
        /// <returns>If all parameters entered and delta is zero or positive
        /// number, it returns a string that contains equations roots.</returns>
        public string Solve()
        {
```

```
        if (areParametersEntered)
        {
            delta = b * b - 4 * a * c;
            if (delta >= 0)
            {
                x1 = (-b + Math.Sqrt(delta) / (2 * a));
                x2 = (-b - Math.Sqrt(delta) / (2 * a));
                return string.Format("X1 = {0}, X2 = {1}",
                    Math.Round(x1,round),
                    Math.Round(x2, round));
            }
            else
                return "Impossible equation";
        }
        else
            return string.Empty;
    }

    /// <summary>
    /// Checks that equation is solvable.
    /// </summary>
    /// <returns>true if delta is zero or positive number; otherwise,
    /// false.</returns>
    public bool isSolvable()
    {
        if (areParametersEntered)
            if (delta >= 0)
                return true;
            else
                return false;
        else
            return false;
    }

    void hasFullParameters()
    {
        if (!double.IsNaN(a) & !double.IsNaN(b) & !double.IsNaN(c))
            areParametersEntered = true;
        else
            areParametersEntered = false;
    }

    /// <summary>
    /// Get or set frist constant of the equation.
    /// </summary>
    public double A
    {
        get { return a; }
        set { if (a!=0) a = value; hasFullParameters(); }
    }

    /// <summary>
    /// Get or set second constant of the equation.
    /// </summary>
    public double B
    {
        get { return b; }
        set { b = value; hasFullParameters(); }
    }
```

```
        /// <summary>
        /// Get or set third constant of the equation.
        /// </summary>
        public double C
        {
            get { return c; }
            set { c = value; hasFullParameters(); }
        }

        /// <summary>
        /// Get delta of the equation
        /// </summary>
        public double Delta
        {
            get { return delta; }
        }

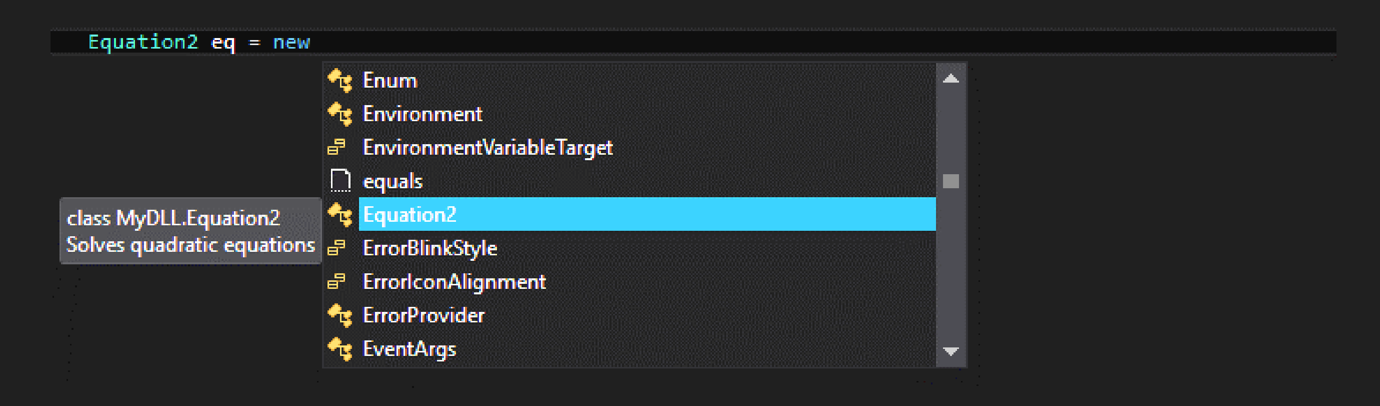
        /// <summary>
        /// Rounds the result to the nearest integer or to the specified number of
        /// fractional digits.</summary>
        public int Round
        {
            get { return round; }
            set { round = value; }
        }
    }
}
```

لاحظ أنني لم أوثّق الطرق أو المتغيرات الخاصة بالمشروع، وثّقت فقط ما سيظهر للمبرمج الذي سيعتمد على مكتبة الأكواد هذه كمرجع. وهنا أريد منك أن تضع بذهنك أننا نقوم بهذا كله للمبرمجين المستخدمين لمكتباتنا، وليس للمستخدمين الذين سيستخدمون البرامج التي يبرمجها المبرمجون الذين استخدموا مكتباتنا (ياربّ ما حدا يدعي علي من هالشرح 😂).

خطوة أخيرة: انتقل لخصائص المشروع Properties، في قسم بناء المشروع Build فعّل تضمين ملف التوثيق XML documentation file مع المشروع، لا تنس القيام بهذه الخطوة عند إضافتك للتوثيقات في ملفات dll الخاصة بك إذا أردت تصديرها للمبرمجين الآخرين (بقيامك بهذه الخطوة فإن ملفًّا من النوع xml سيتم إنشاؤه بجانب ملف dll الخاص بك، لذلك يجب نسخه مع ملف dll عند توزيعه على المبرمجين، فهو يحوي التوثيقات!!).

الآن نقّح Debug المشروع MyDLL ثم عد لمشروع النوافذ ولاحظ:

عند استنساخ الفئة:



عند إدخال الوسطاء:

```
Equation2 eq = new Equation2(
```

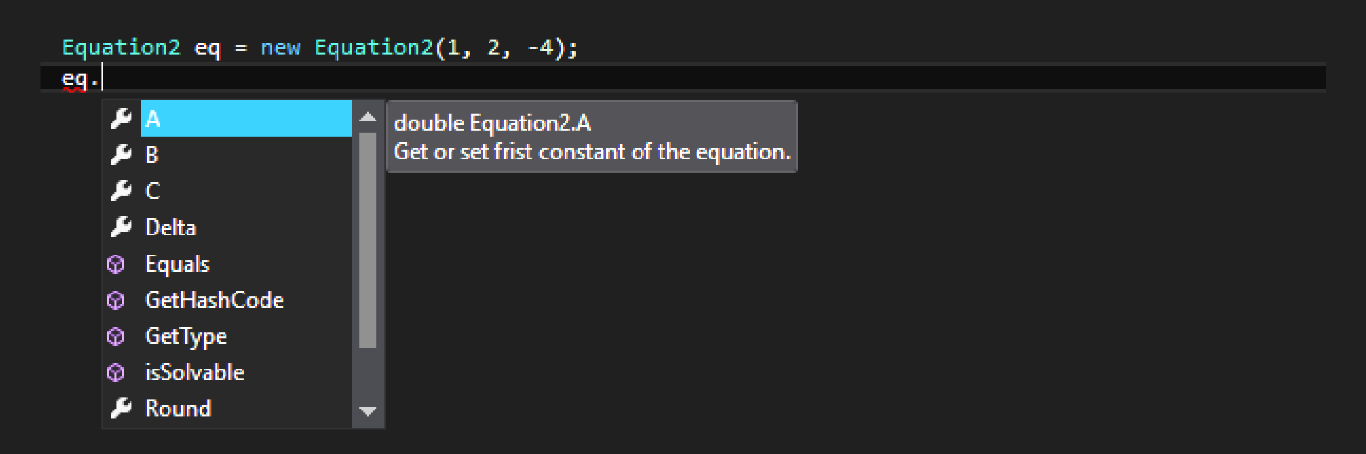
▲ 2 of 2 ▼ Equation2.Equation2(**double A**, double B, double C)
Creates an instance to Equation2 class with parameters.
A: *First constant.*

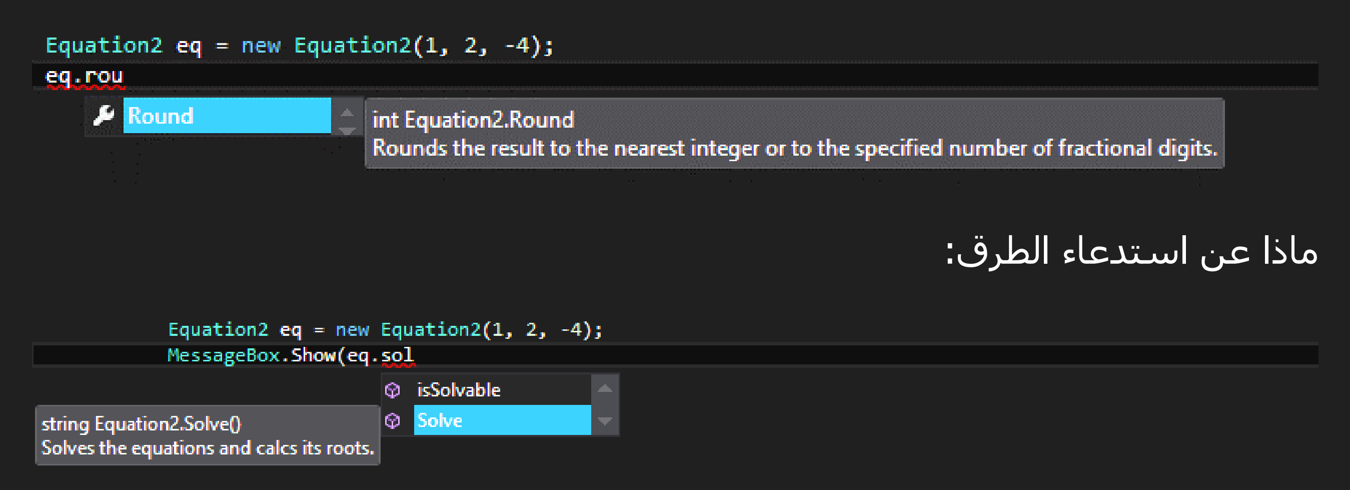
لاحظ الوسيط الثاني:

```
Equation2 eq = new Equation2(1,
```

▲ 2 of 2 ▼ Equation2.Equation2(double A, **double B**, double C)
Creates an instance to Equation2 class with parameters.
B: *Second constant.*

أما لو أردنا الوصول لخصائص الكائن:





ماذا عن استدعاء الطرق:

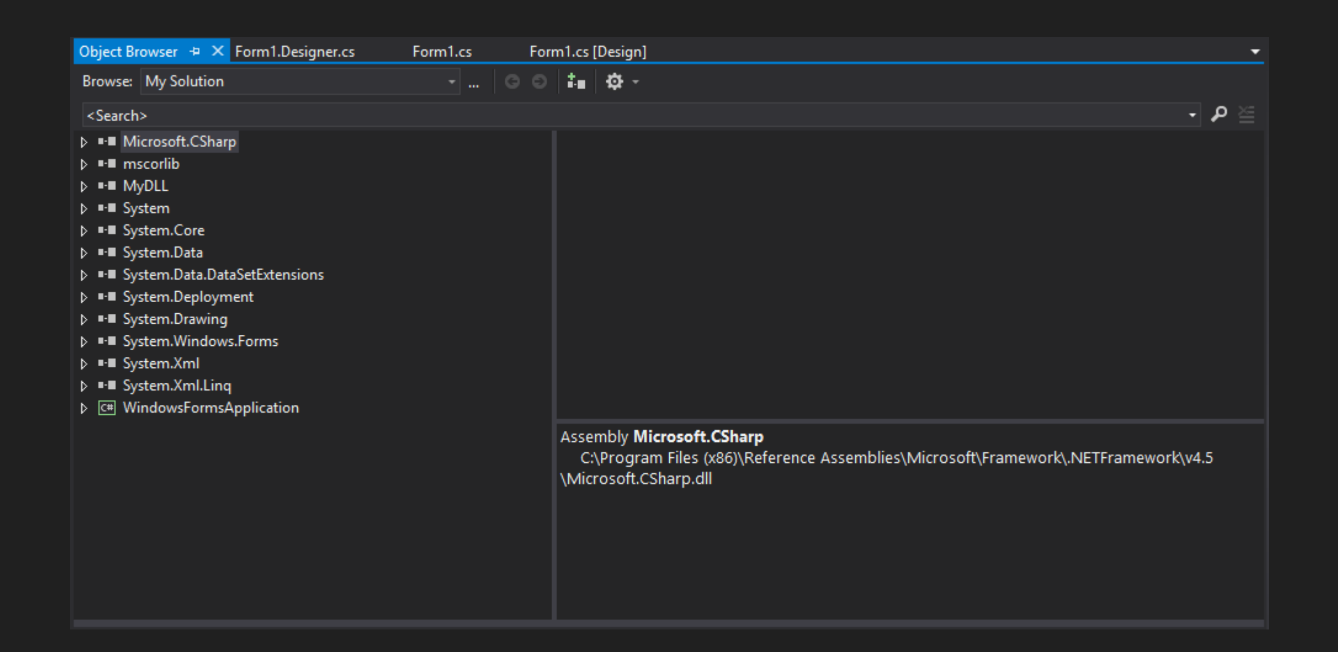
هل لازلت تبحث عن غاية وجود الملاحظات الكثيرة في كود المصمم؟؟

مستعرض الكائنات Objects Browser

يمكنك الحصول على معلومات كثيرة عن الكائنات البرمجية الخاصة بالفئات ومكتبات الارتباط الحيوي dll وغيرها من مصادر مشروعك عن طريق مستعرض الكائنات هذا، ويمكن الوصول إليه عن طريق القائمة View ثم Object browser.



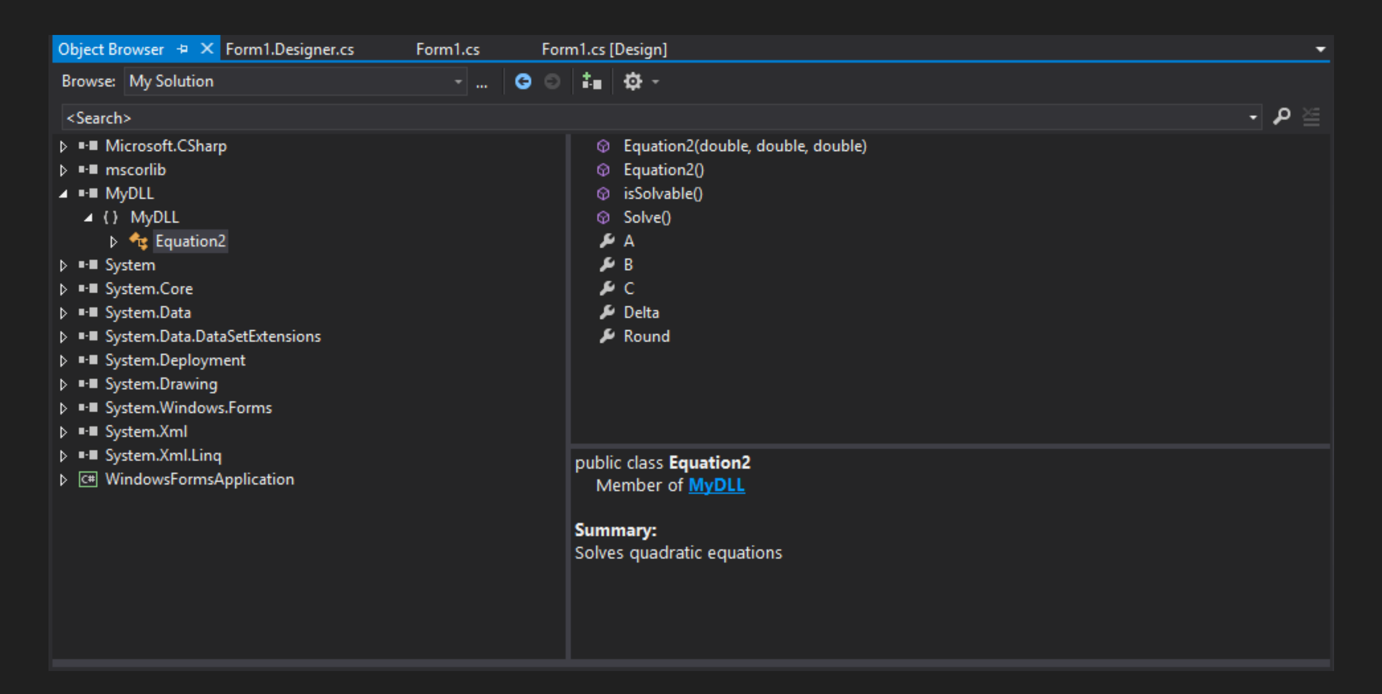
يمكن الوصول لمستعرض الكائنات عبر الاختصار Ctrl+Alt+J.



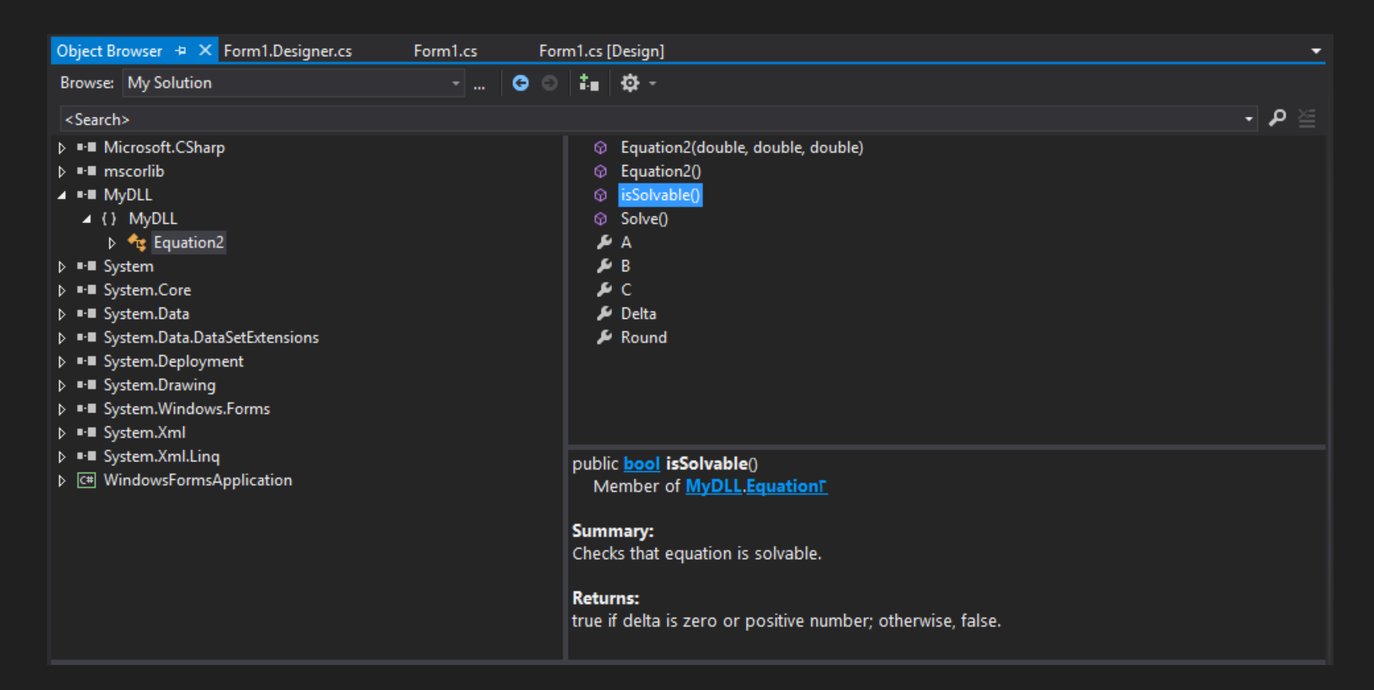
المحتويات الموجودة في الصورة الأخيرة هي مراجع المشروع والمشروع نفسه، كما أننا أضفنا مكتبة MyDLL من الفقرة السابقة (إذا أضفت مراجع أخرى ستظهر هنا، وإذا حذفت غيرها ستُزال من هنا).

تجد في نافذة مستعرض الكائنات – الصورة السابقة نفسها – ثلاثة أقسام: قسم يحوي مجالات الأسماء والفئات، وقسم يحوي محتويات الفئات (الخصائص والطرق والأحداث)، وقسم يحوي شرحًا عما هو تحت التركيز (في الصورة السابقة التركيز على مجال الأسماء Microsoft.CSharp)

لنبدأ مثلًا مع مجال الأسماء MyDLL، ونوسّع محتوياته:



لاحظ المحتوى، والوصف. ماذا لو وضعنا التركيز على تابع ما:



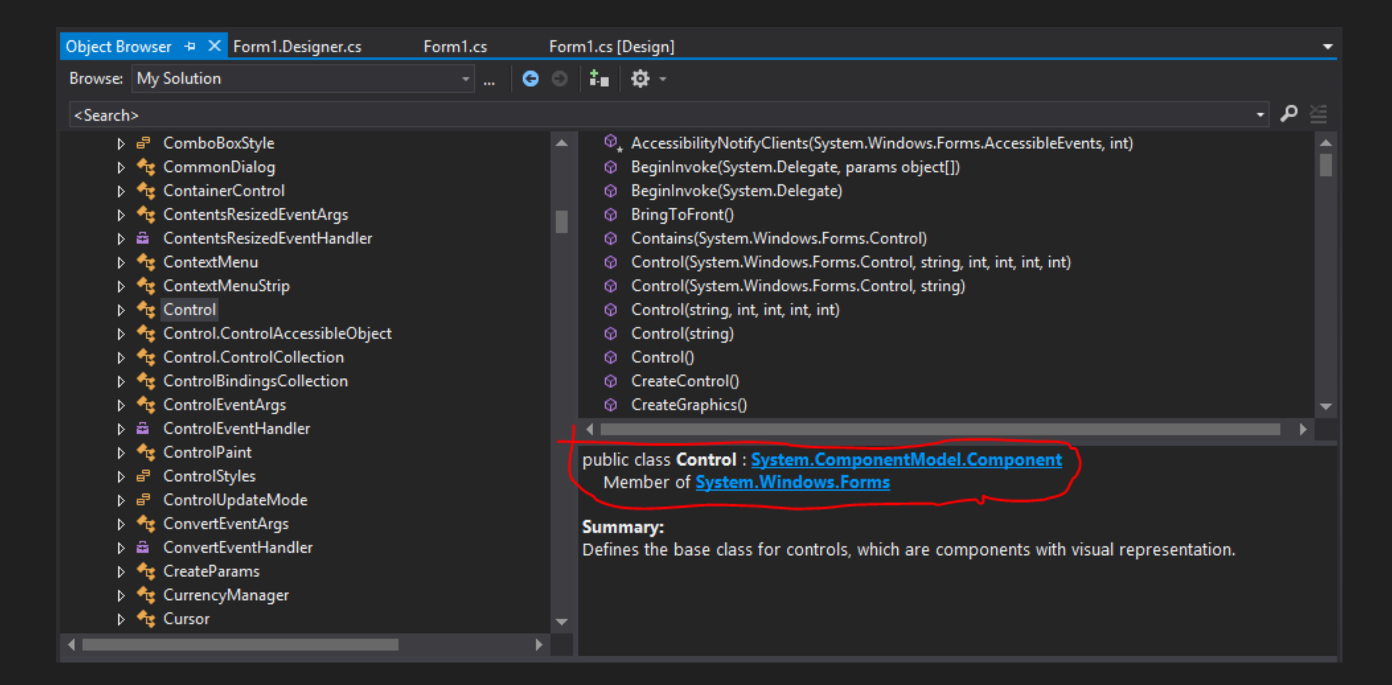
لاحظ أن الوصف هو تماما كما وضعناه في فقرة **التوثيق XML Documentation**، وأن وسطاء التوابع والإجراءات يتم توضيحها بشكل كامل من حيث نوعها وعددها.

انتقل مثلًا إلى System.Windows.Forms ثم Control، ستجد فيها مجموعة من الطرق والأحداث والخصائص التي توجد في أغلب الأدوات. هل تذكر فقرة مبادئ OOP الأساسية في بداية هذا الفصل، والتي تحدثنا في نهايتها عن الفئات المجردة التي ننشئها ليس بغية اشتقاقها وإنما بغية إنشاء مفاهيم مشتركة بين الفئات المختلفة، فنقوم بوراثة هذه الفئة في كل فئة ستحمل هذه المفاهيم؟ صحيح أن الفئة Control ليست فئة مجرّدة – وهذا حتى نتمكن من إنشاء كائنات من نوعها – ولكنها تعمل كعمل الفئات المجرّدة المقصودة بفقرة المبادئ الأساسية التي أشرنا إليها.

تأمل محتويات الفئة Control ولاحظ أن طرقها وأحداثها وخصائصها هي أشياء توجد في أكثر الأدوات، وعوضًا عن إنشاء هذه الأشياء في كل أداة بأداتها نعتمد على مبدأ الوراثة. أذكر مرّةً قرأت في كتاب **فيجوال بيسك للجميع، نحو برمجة كائنية التوجه** للأستاذ الكبير تركي العسيري: "من الاجراء السابق يتضح لنا جمال، قوة، ابداع، مرونة، فن، وسحر مبدأ تعدد الواجهات فالفئة TextBox لها واجهة اخرى باسم Control تحتوي على

الطريقة Move حالها كحال جميع الادوات الاخرى. الواجهة Control هي عبارة عن فئة لكنها لا تحتوي على اية اكواد، لذلك تسمى فئة المجردة Abstract Class Control وتحتوي على واجهة Interface، فحتى تستطيع ان تحقق مبدأ تعدد الواجهات لابد من وجود فئة مجردة والتي تعرف الواجهة للفئات الأخرى منها" [1]. وواضح أنه يتحدث عن نفس موضوعنا، إلا أنه على أيام VB6 كانت الفئة Control مجردة، أما في #C فلا.

وقد كنت يومها حديث عهد بالبرمجة، فنظرت إلى كلامه وبحثت عن الجماع والقوة والإبداع والمرونة والفن والسحر في الأسماء التي سماها هو ومايكروسوفت 😇! ولكنني الآن مؤمن بكلامه 😌.

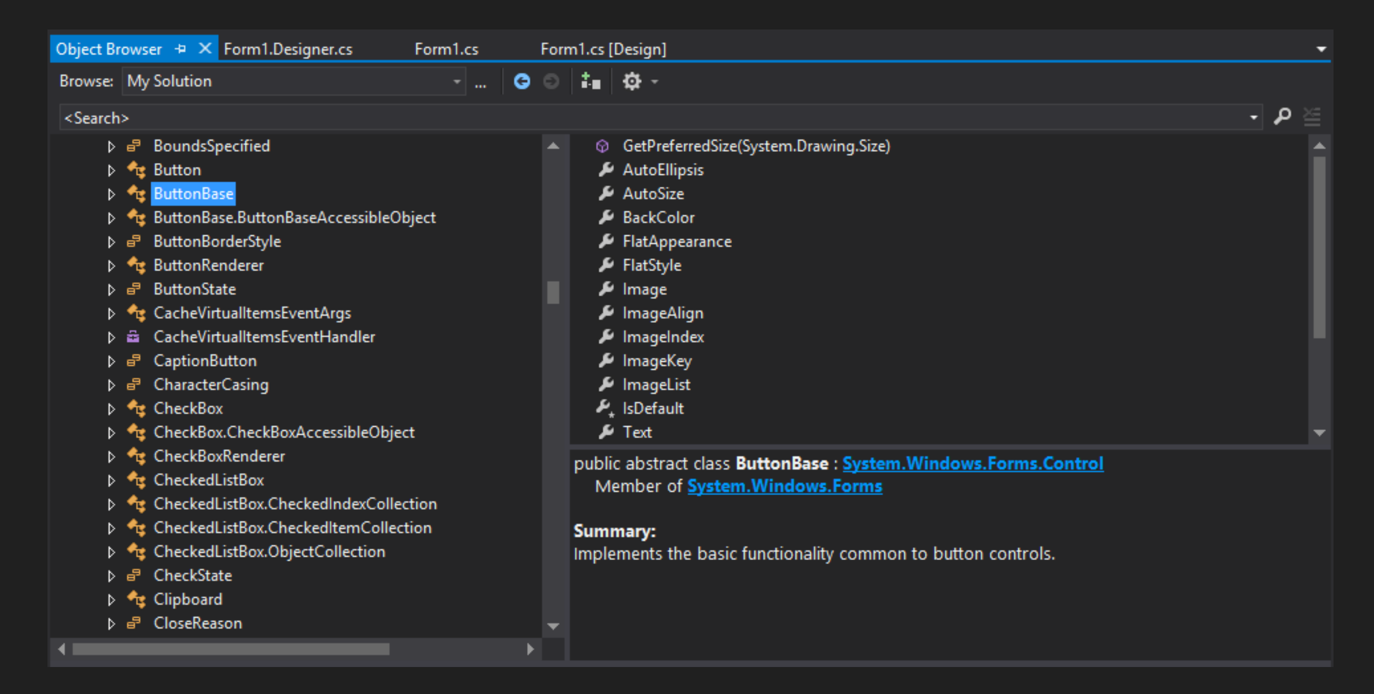


من الصورة الأخيرة يتضح لنا أن Control ليست مجردة لعدم وجود كلمة abstract في تعريفها في الوصف، كما أنها ترث من System.ComponentModel.Component وتقع في مجال الأسماء System.Windows.Forms.

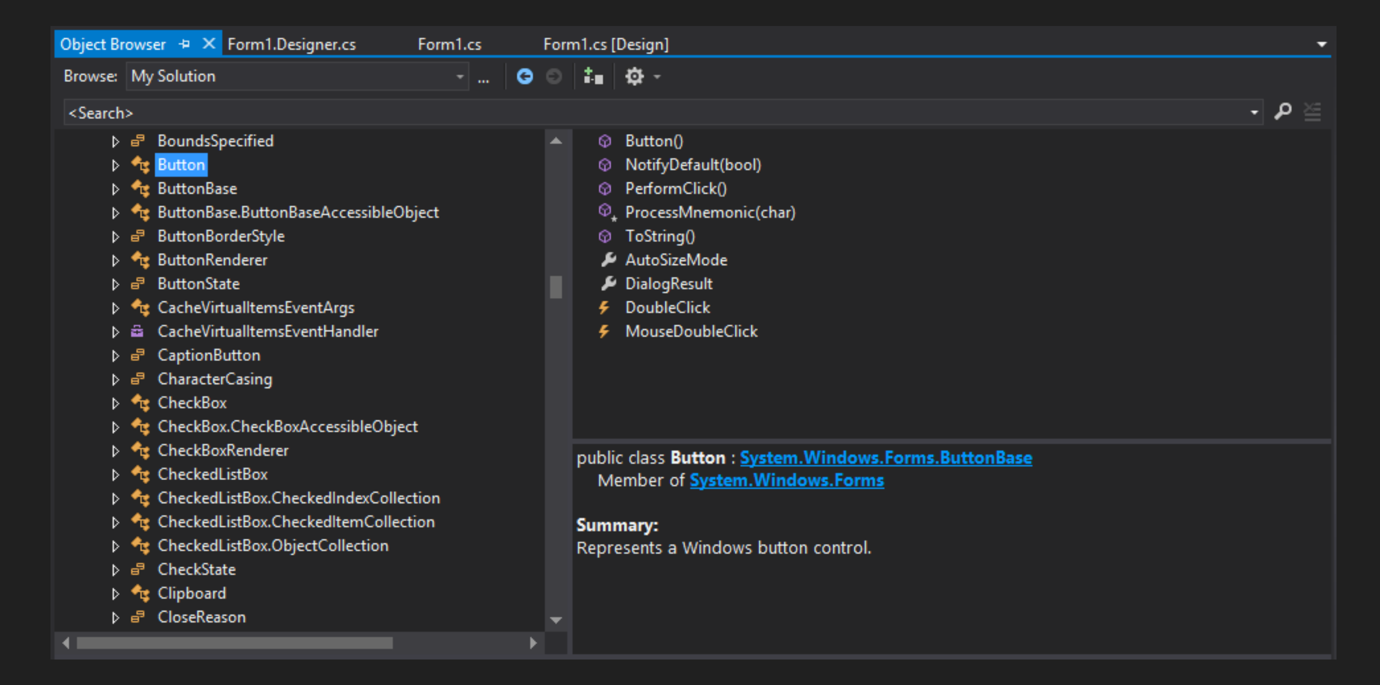
لاحظ أيضًا أنه قيل عن هذه الفئة أنها تعرّف الفئة الأم للأدوات، والتي هي عبارة عن أدوات تستخدم في مشاريع النوافذ (يُقصد من هذا أدوات مثل Button وTextBox وغيرها).

[1] انظر كتاب "فيجوال بيسك للجميع، نحو برمجة كائنية التوجه" لـ تركي العسيري (ص 176).

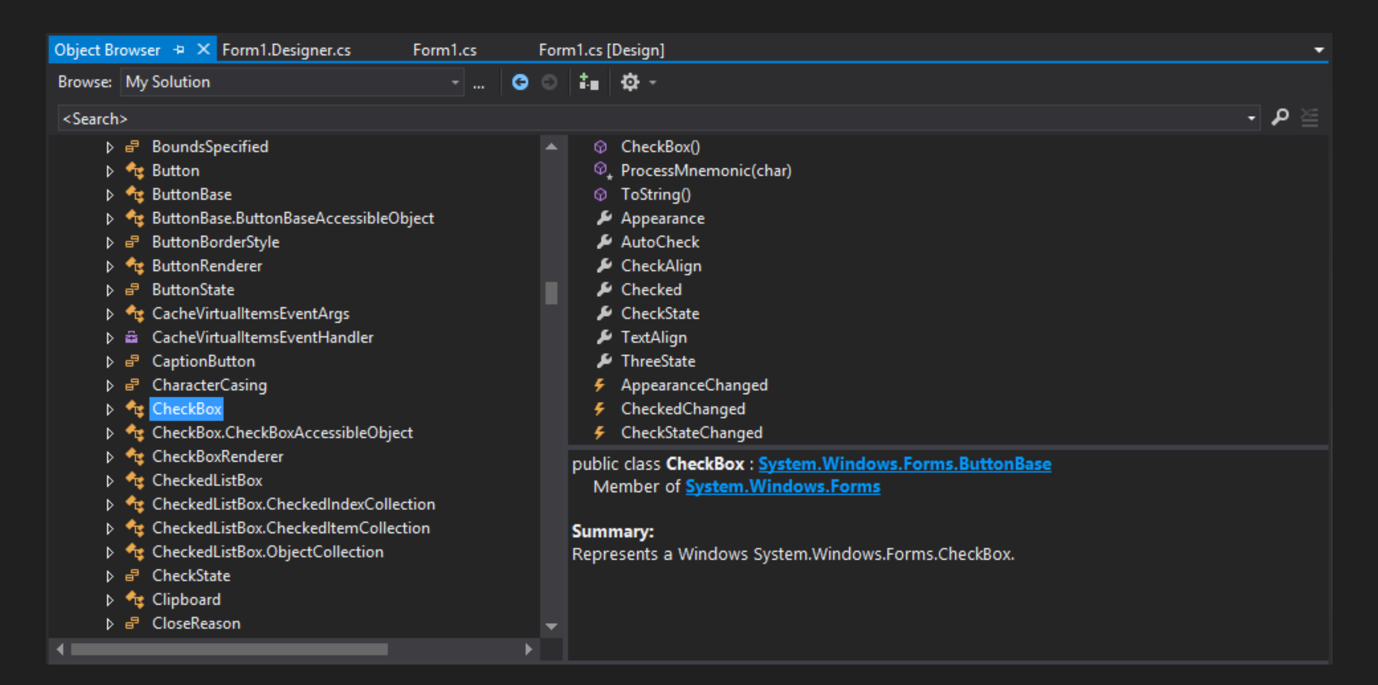
انتقل إلى الفئة ButtonBase في نفس مجال الأسماء، لاحظ أنها مجردة وأنها ترث من الفئة Control أي كل ما يوجد في Control يوجد في هذه الفئة:



انتقل إلى Button، ولاحظ أن مكوناتها محدودة جدًّا، في حين أن الأدوات من هذا النوع تحوي الكثير والكثير من الخصائص والطرق والأحداث، وهذا دليل على أن هذه الفئة لا بد لها من الوراثة من فئات أخرى حتى تحصل على تفاصيلها:



انتقل إلى CheckBox ولاحظ أنها أيضا ترث من ButtonBase، وقد يخطر على بالك أنها لا تحوي أزرارًا فلم ترث من فئة تمثل الأزرار؟ في الواقع من الممكن أن تجعل أدوات هذا النوع على شكل زر، كما أن الشكل العادي لها يأخذ أوامر بالنقر عليه، فبالتالي هو زر حتى لو لم يكن شكله كذلك:



مما سبق، تجلّت أهمية مستعرض الكائنات، وكيفية الاستفادة منه، وكيفية تحليل الكائنات وفهمها وفهم علاقاتها مع الكائنات الأخرى. ليس هذا فحسب، بل وجدنا كيف أن أدوات مايكروسوفت كائنية التوجه بامتياز، وتحقق جميع مبادئ OOP، وتبدأ بفئات عامة تحوي أعضاءً كثيرة مشتركة بين السواد الأعظم من الأدوات، وتنتهي بفئات تمثل الأدوات بحد ذاتها، وتحوي أعضاء محدودة، تشكّلها، وتميز كل أداة عن غيرها.

كما أن الفيجوال ستوديو يحوي أدوات أخرى تستحق منك البحث عنها والاطلاع عليها. وهذا كله ستحتاجه بطريقة غير مباشرة في طيّات هذا الكتاب، وعلى اعتبار أن محور الكتاب يهتم بالأدوات وإنشاءها فإنك تحتاج هذه الأدوات – مثل مستعرض الكائنات – لفهم الأدوات الموجودة أصلًا.

الفصل الثالث – تقنيات دعم البرامج

تحتاج البرامج – الكبيرة منها خصوصًا – إلى وسائل وتقنيات تدعم عملها، كأدوات تثبيتها وإلغاء تثبيتها ومراقبتها وما إلى ذلك. وقد ناقشنا في كتابنا الأول كلًّا من الرجستري وموجه الأوامر، إذ فيهما دعم كبير للبرامج. وفي هذا الفصل سنناقش تقنيات أخرى.

خدمات ويندوز Windows Services

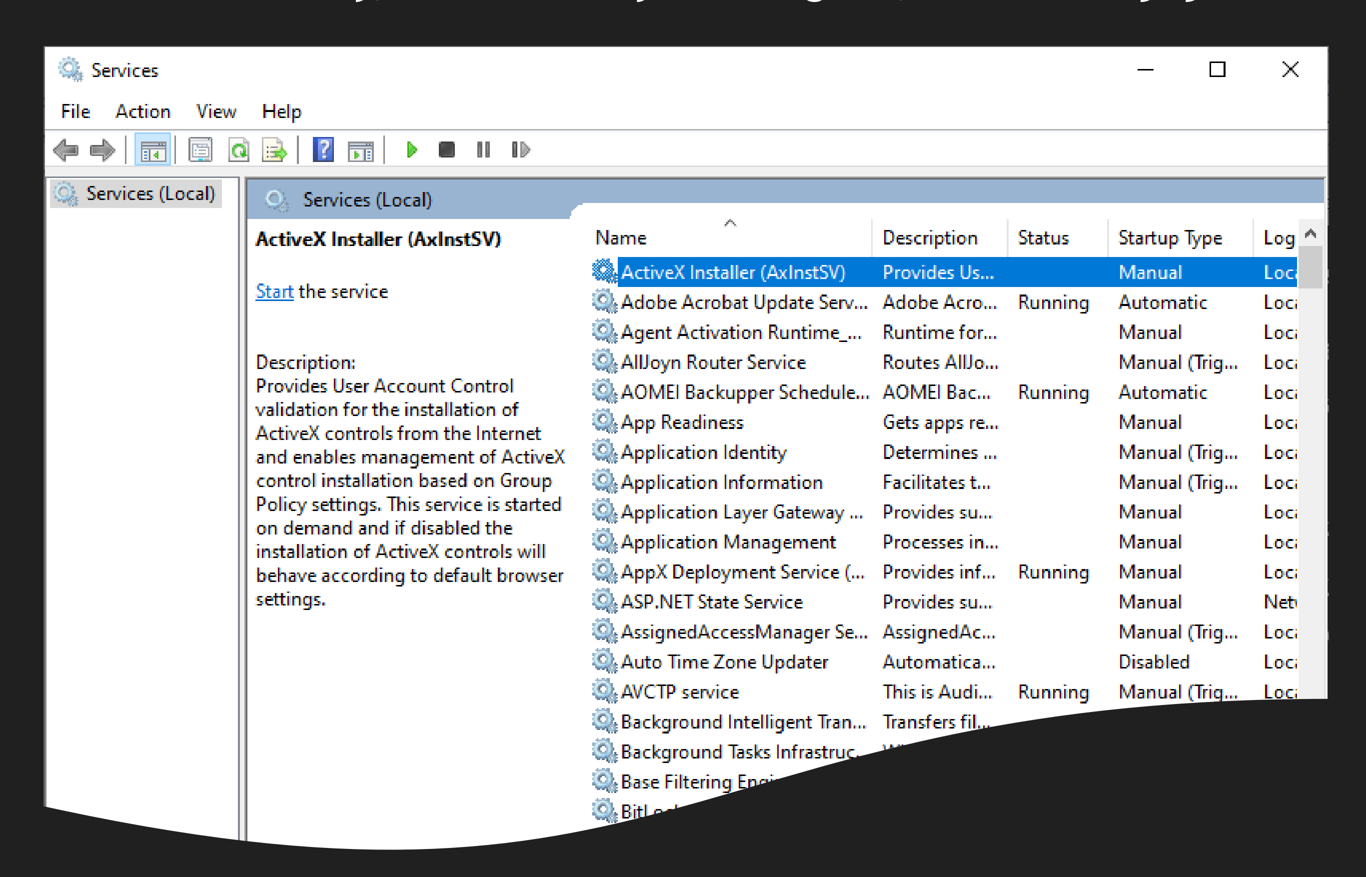
خدمات ويندوز هي ملفات تنفيذية exe عادية لكنها تعمل لفترات طويلة، بدءًا من لحظة تشغيل نظام التشغيل، ولا تتطلب تدخلًا في عملها من قبل المستخدم، إذ إنها تعمل بالخلفية Background ولا تتضمن واجهة مستخدم User Interface؛ وعليه فإنها لا تحتاج

وجود حساب مستخدم نشط، فيمكنها العمل حتى لو لم يتم تسجيل الدخول إلى أي مستخدم. [1]

يمكن لملف تنفيذي exe أن يحوي أكثر من خدمة ويندوز، لكنه يجب أن يحوي فئة ServiceInstaller [2] خاصة بكل خدمة. هذه الفئة تقوم بتسجيل الخدمة في النظام. الطريقة main تحدد الخدمات التي ستعمل. الفهرس (المجلد) الحالي لأي خدمة ويندوز ليس مسار الملف التنفيذي الحاوي على الخدمة وإنما مسار مجلد النظام. [3]

تستخدم خدمات ويندوز لإدارة العمليات طويلة الأمد والتي لا علاقة للمستخدم بها لا من قريب ولا من بعيد، كالتحقق من التحديثات والتعامل مع الأجهزة المتصلة بالكمبيوتر مثل البلوتوث ومراقبة الملفات.

يمكنك الوصول للخدمات المثبّتة من خلال الأمر services.msc عبر الأداة Run:



[1] ميكروسوفت – الفئة ServiceBase
https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase?view=dotnet-plat-ext-3.1
[2] سنرى الفئة ServiceInstaller في فقرة قادمة.
[3] المصدر السابق (ميكروسوفت – الفئة ServiceBase).



يمكنك الوصول للأداة Run من خلال Win} + R}.

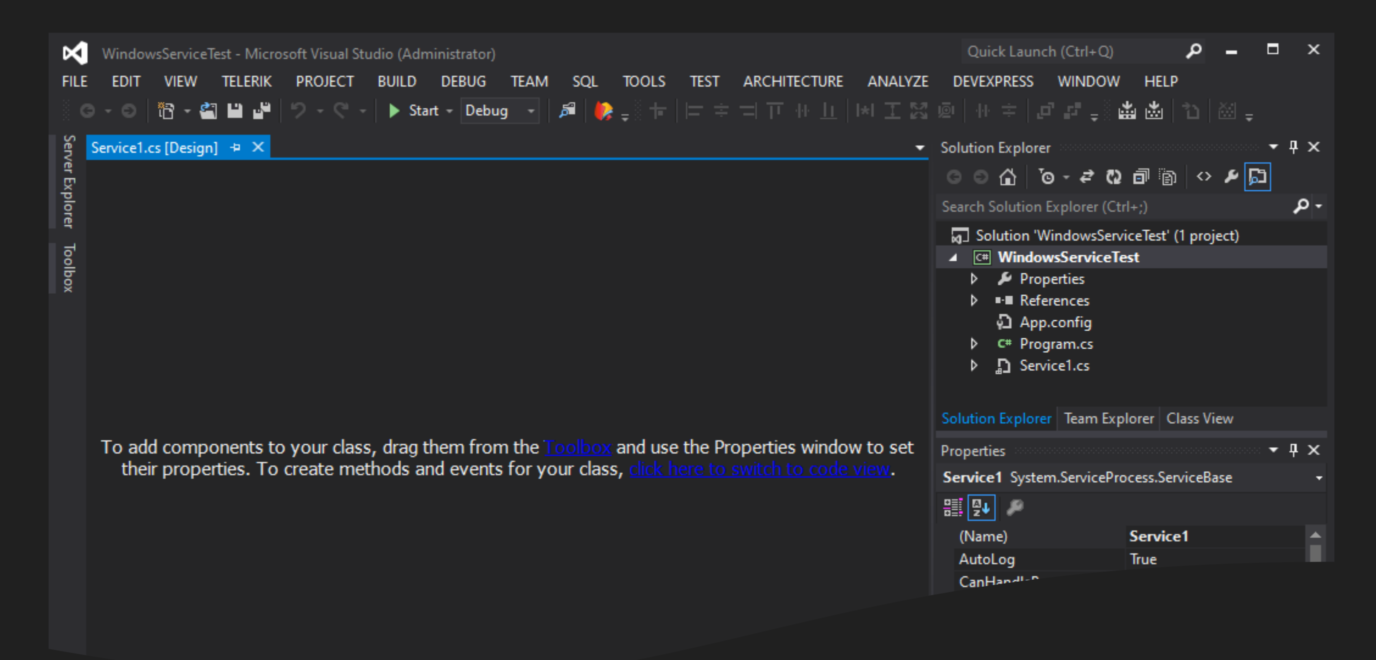
من خلال الأداة Services يمكنك التحكم بالخدمات، كإيقافها أو تشغيلها، فلا يمكنك ذلك من خلال ملفات exe الحاوية على هذه الخدمات على اعتبار أنها لا تحوي واجهة مستخدم (عند تشغيل تطبيق خدمة ويندوز ستحصل على رسالة خطأ شبيهة بتلك التي تحصل عليها عند محاولة تشغيل مشروع مكتبة فئات Class Library).



يمكنك الوصول للأداة Services أيضًا من خلال الأدوات الإدارية Administrative Tools وذلك من لوحة التحكم Control Panel. كما يمكن ذلك من خلال موجّه الأوامر cmd بكتابة الأمر services.msc.

إنشاء خدمة ويندوز

من القائمة File، اختر New ثم Project، ثم اختر مشروعًا من النوع Windows Services وسمه WindowsServiceTest:



لاحظ عدم وجود واجهة مستخدم، فمشاريع خدمات ويندوز أشبه بمشاريع مكتبات DLL.
انتقل لنافذة محرر الكود ولاحظ الكود المولّد تلقائيًّا:



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Linq;
using System.ServiceProcess;
using System.Text;
using System.Threading.Tasks;

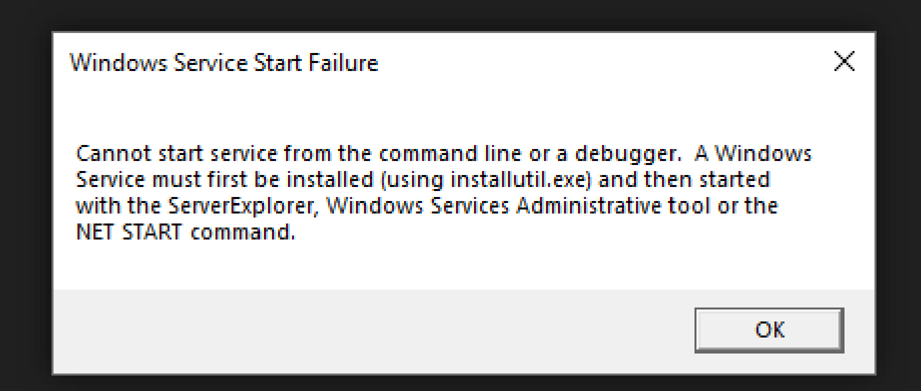
namespace WindowsServiceTest
{
    public partial class Service1 : ServiceBase
    {
        public Service1()
        {
            InitializeComponent();
        }

        protected override void OnStart(string[] args)
        {
        }

        protected override void OnStop()
        {
        }
    }
}
```

لاحظ أن القسم الثاني من الكود فيه إجراءان واضح من اسميهما أنهما يُستدعيان عند بدء تشغيل الخدمة وعند إيقافها فقط.

عند تشغيل خدمة ما، فإن النظام يحدد الملف التنفيذي المتضمن للخدمة، وينفذ الطريقة OnStart. كما أنه لا يمكنك تشغيل الملف التنفيذي الحاوي على الخدمة، إذ إنه يمكن الوصول للخدمة (عند إيقافها وتشغيلها مثلًا) من خلال الأداة Services. وإذا كنت فضوليًّا بما يكفي لتشغيل الملفات التنفيذية للخدمات فإنك ستحصل على رسالة الخطأ هذه:

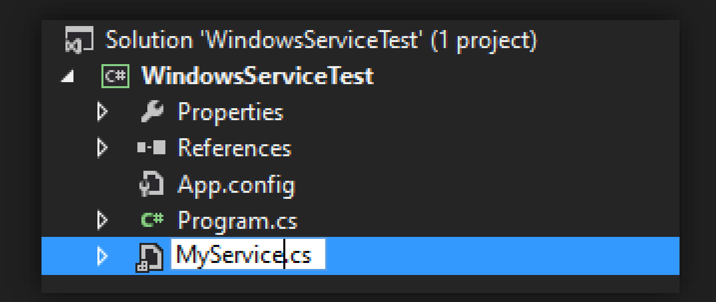


الفئة ServiceBase

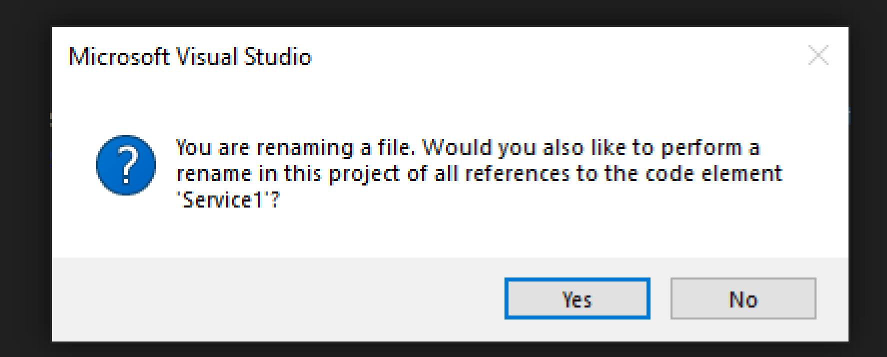
خدمات ويندوز Windows Services هي فئات مشتقة من الفئة ServiceBase، والتي تزود خدماتك بمجموعة من الخصائص والطرق والأحداث المفيدة. بشكل افتراضي فإن كود أي خدمة ويندوز سيحوي على الإجرائين OnStart وOnStop (بعد إعادة تعريفهما)، يمكنك أيضًا إعادة تعريف Override الإجرائين OnPause وOnContinue إن كانت خدماتك قابلة للتوقف المؤقت والاستمرار.

تغيير اسم خدمة ويندوز

من المهم تحديد اسم الخدمة، وهو ما سيظهر في الأداة Services. من متصفح المشروع Solution Explorer غير اسم فئة الخدمة لـ MyService مثلًا:



ستظهر رسالة تسألك فيما إذا كنت ترغب بتغيير أكواد مشروعك ليتماشى مع الاسم الجديد لفئة الخدمة، انقر على نعم:



انقر على منطقة التصميم – الخالية – لتظهر خصائص الخدمة في صندوق الخصائص. غير اسم الخدمة ServiceName لـ "MyService by Eng27" أو ما شابه ذلك.

إضافة وظائف لخدمة ويندوز

إن المهام المسندة لوظائف ويندوز – كما أسلفنا – تكون روتينية ودورية وتعمل لفترات طويلة، وخير ما نشرح به مشاريع خدمات ويندوز هو مثال تطبيقي عملي، وهذا المثال فيه فكرة مأخوذة من كتاب "برمجة أطر عمل .NET" للأستاذ تركي العسيري وهي مراقبة الملفات. يمكن مراقبة الملفات من خلال الفئة FileSystemWatcher والتي تعطيك إمكانية مراقبة الملفات ومعرفة ما إذا تم إنشاؤها أو حذفها أو إعادة تسميتها أو التعديل عليها.



يمكنك استخدام الفئة FileSystemWatcher – التابعة لمجال الأسماء System.IO – في مشاريع النوافذ أيضًا، لكن على اعتبارها تقوم بعمليات طويلة الأمد – ولا حاجة للمستخدم بالتفاعل معها – فاستخدامها ضمن مشروع خدمات ويندوز أفضل.

عدل الكود ليصبح كما يلي:

```
using System;
using System.IO;
using System.ServiceProcess;

namespace WindowsServiceTest
{
```

```
public partial class MyService : ServiceBase
{
    FileSystemWatcher fsw = new FileSystemWatcher();
    string path;
    string filepath;

    public MyService()
    {
        InitializeComponent();
        fsw.Path = "D:\\";
        fsw.EnableRaisingEvents = true;
        fsw.IncludeSubdirectories = true;
        fsw.Deleted += fsw_Deleted;
        fsw.Created += fsw_Created;
        fsw.Changed += fsw_Changed;
        fsw.Renamed += fsw_Renamed;
    }

    protected override void OnStart(string[] args)
    {
        WriteToFile("Service is started at " + DateTime.Now);
    }

    protected override void OnStop()
    {
        WriteToFile("Service is stopped at " + DateTime.Now);
    }

    void fsw_Renamed(object sender, RenamedEventArgs e)
    {
        if (e.FullPath != filepath)
            WriteToFile(e.ChangeType + ": " +
                e.FullPath +
                " (Name was " +
                e.OldName +
                ") at " +
                DateTime.Now);
    }

    void fsw_Changed(object sender, FileSystemEventArgs e)
    {
        //if (e.FullPath != filepath)
        //    WriteToFile(e.ChangeType+": "+e.FullPath+" at "+DateTime.Now);
        //ألغيت هذا الكود لأن هناك احتمال وجود الملف الذي سيتم كتابة سجل المراقبة فيه، في المسار الذي سنراقبه
        //وعندها ستحصل على حلقة متداخلة تتكرر إلى ما لا نهاية
        //إذا كنت متأكدا أن الملف الذي ستكتب به سجل المراقبة غير موجود ضمن المجلد المراقب ففعل هذا الكود
    }

    void fsw_Created(object sender, FileSystemEventArgs e)
    {
        if (e.FullPath != filepath)
            WriteToFile(e.ChangeType +
                ": " +
                e.FullPath +
                " at " +
                DateTime.Now);
    }
```

```
        void fsw_Deleted(object sender, FileSystemEventArgs e)
        {
            if (e.FullPath != filepath)
                WriteToFile(e.ChangeType +
                    ": " +
                    e.FullPath +
                    " at " +
                    DateTime.Now);
        }

        public void WriteToFile(string s)
        {
            path = AppDomain.CurrentDomain.BaseDirectory + "\\Logs";
            if (!Directory.Exists(path))
                Directory.CreateDirectory(path);

            filepath = path +
                "\\ServiceLog_" +
                DateTime.Now.Date.ToShortDateString().Replace('/', '-') +
                ".txt";

            if (!File.Exists(filepath))
                using (StreamWriter sw = File.CreateText(filepath))
                {
                    sw.WriteLine(s);
                }
            else
                using (StreamWriter sw = File.AppendText(filepath))
                {
                    sw.WriteLine(s);
                }
        }
    }
}
```

تم حفظ سجل المراقبة ضمن ملف بجوار الملف التنفيذي الذي يمثل مشروعك، وهو ما لا أفضله، فالبيانات الكثيرة والتي فيها شيئا من الروتينية والتكرار والكمية يفضل حفظها ضمن قاعدة بيانات. ولكن حتى لا تتشعب الأفكار اقتصرت على حفظ البيانات ضمن ملف نصي لعرض النتيجة لا أكثر.

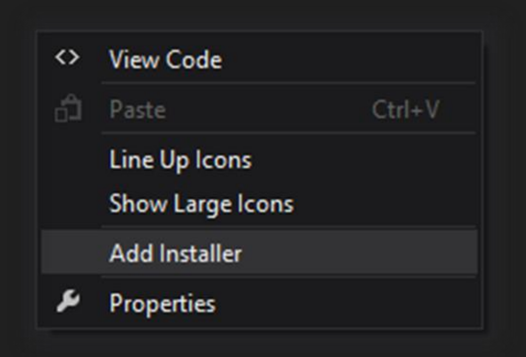
لاحظ أيضًا أن المجلد الذي سيتم مراقبته هو القرص D، يمكنك تخزين بعض البيانات مثل المجلد المراد مراقبته أو مجلد تخزين السجلات أو غيرها من البيانات ضمن ملف ما أو قاعدة بيانات، بحيث تديرها من برنامج آخر تصممه لهذا الغرض. بمعنى آخر، يمكنك إنشاء برامج تنفيذية تضبط فيها بعض الخيارات التي ترغب من خدمات ويندوز الخاصة بك أن تقوم بها.

الجدير بالذكر أن الإجراءين OnStart وOnStop لن يتم تنفيذهما عند تشغيل النظام وإيقافه، وإنما عند تشغيل الخدمة وإيقافها، لذلك فإنك لن ترى عبارة "تم بدء الخدمة في الوقت كذا" و"تم إيقاف الخدمة في الوقت كذا" في كل مرة توقف أو تشغّل فيها نظام التشغيل.

إنشاء مثبت لخدمة ويندوز

لا يمكنك تثبيت (تركيب) الخدمة يدويًّا، وإنما عليك ذلك من خلال أدوات جاهزة. عليك أولًا إضافة أداة مثبت الخدمات ServiceInstaller من الفيجوال ستوديو في مشروعك، ثم تثبيت الخدمة في ويندوز من خلال أداة InstallUtil.exe الملحقة بنظام التشغيل.

انتقل لمنطقة التصميم – الخالية – وانقر باليمين ثم اختر Add Installer:



ستحصل على أداتين جديدتين:



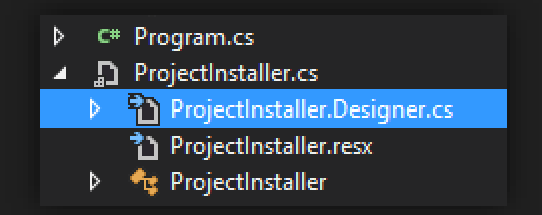
انقر بالزر الأيمن على أي من الأداتين – أو على المنطقة الخالية – ثم اختر View Code لتنتقل لمحرر الكود والذي يفترض أن يكون بالشكل:



```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration.Install;
using System.Linq;
using System.Threading.Tasks;

namespace WindowsServiceTest
{
    [RunInstaller(true)]
    public partial class ProjectInstaller : System.Configuration.Install.Installer
    {
        public ProjectInstaller()
        {
            InitializeComponent();
        }
    }
}
```

انتقل للمصمم، وذلك من خلال مستعرض المشروع Solution Explorer:





يمكنك الانتقال لمصمم أي نافذة بإيقاف مؤشر الفأرة على الإجراء InitializeComponent ثم الضغط على {F12}، والذي ينقلك لمكان تواجد هذا الإجراء، إذ إن هذا الإجراء يتواجد بشكل افتراضي في المصمم.

عدل الإجراء ليصبح بالشكل التالي:

```
private void InitializeComponent()
{
    this.serviceProcessInstaller1 = new System.ServiceProcess.ServiceProcessInstaller();
    this.serviceInstaller1 = new System.ServiceProcess.ServiceInstaller();
    //
    // serviceProcessInstaller1
    //
    this.serviceProcessInstaller1.Account = System.ServiceProcess.ServiceAccount.LocalSystem;
    this.serviceProcessInstaller1.Password = null;
    this.serviceProcessInstaller1.Username = null;
    //
    // serviceInstaller1
    //
    this.serviceInstaller1.Description = "MyService By Eng27";
    this.serviceInstaller1.DisplayName = "MyService By Eng27";
    this.serviceInstaller1.ServiceName = "MyService by Eng27";
    //
    // ProjectInstaller
    //
    this.Installers.AddRange(new System.Configuration.Install.Installer[] {
    this.serviceProcessInstaller1,
    this.serviceInstaller1});
}
```

الآن، قم ببناء مشروعك ليتم تأليف ملف تنفيذي exe يمثل مشروعك بما فيه (ومع أنه من النوع exe إلا أنه لا يمكن تشغيله). ثم افتح موجّه الأوامر كمسؤول وانتقل من خلاله للمسار "C:\Windows\Microsoft.NET\Framework\v4.0.30319" ثم اكتب الأمر:

InstallUtil.exe "مسار الملف التنفيذي"

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.18362.356]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Eng27>cd C:\Windows\Microsoft.NET\Framework\v4.0.30319   1

C:\Windows\Microsoft.NET\Framework\v4.0.30319>installutil.exe "E:\C#\KG\WindowsServiceTest\WindowsServiceTest\bin\Debug\WindowsServiceTest.exe"  2
Microsoft (R) .NET Framework Installation utility Version 4.8.3752.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Running a transacted installation.

Beginning the Install phase of the installation.
See the contents of the log file for the E:\C#\KG\WindowsServiceTest\WindowsServiceTest\bin\Debug\WindowsServiceTest.exe assembly's progress.
The file is located at E:\C#\KG\WindowsServiceTest\WindowsServiceTest\bin\Debug\WindowsServiceTest.InstallLog.
Installing assembly 'E:\C#\KG\WindowsServiceTest\WindowsServiceTest\bin\Debug\WindowsServiceTest.exe'.
Affected parameters are:
   logtoconsole =
   logfile = E:\C#\KG\WindowsServiceTest\WindowsServiceTest\bin\Debug\WindowsServiceTest.InstallLog
   assemblypath = E:\C#\KG\WindowsServiceTest\WindowsServiceTest\bin\Debug\WindowsServiceTest.exe
No public installers with the RunInstallerAttribute.Yes attribute could be found in the E:\C#\KG\WindowsServiceTest\WindowsServiceTest\bin\Debug\WindowsServiceTest.exe
assembly.

The Install phase completed successfully, and the Commit phase is beginning.
See the contents of the log file for the E:\C#\KG\WindowsServiceTest\WindowsServiceTest\bin\Debug\WindowsServiceTest.exe assembly's progress.
The file is located at E:\C#\KG\WindowsServiceTest\WindowsServiceTest\bin\Debug\WindowsServiceTest.InstallLog.
Committing assembly 'E:\C#\KG\WindowsServiceTest\WindowsServiceTest\bin\Debug\WindowsServiceTest.exe'.
Affected parameters are:
   logtoconsole =
   logfile = E:\C#\KG\WindowsServiceTest\WindowsServiceTest\bin\Debug\WindowsServiceTest.InstallLog
   assemblypath = E:\C#\KG\WindowsServiceTest\WindowsServiceTest\bin\Debug\WindowsServiceTest.exe
No public installers with the RunInstallerAttribute.Yes attribute could be found in the E:\C#\KG\WindowsServiceTest\WindowsServiceTest\bin\Debug\WindowsServiceTest.exe
assembly.
Remove InstallState file because there are no installers.

The Commit phase completed successfully.

The transacted install has completed.

C:\Windows\Microsoft.NET\Framework\v4.0.30319>
```



الغاية من الانتقال للمسار المذكور هو حتى يصبح الفهرس (المجلد) الذي يتواجد فيه موجه الأوامر في لحظة تنفيذ الأمر بجوار الملف التنفيذي الذي سيتم تنفيذ أوامره، إذ إن هذا الأمر الذي تم تنفيذه ليس من الأوامر الداخلية في موجه الأوامر، وبالتالي يجب تضمين الملف التنفيذي المصدر الذي يحوي هذا الأمر.

يمكنك إنشاء أداة تقوم بالاعتماد على الملفات الدفعية لتنفيذ هذه المهمة، وهو ما سنتناوله في الفقرة "أتمتة عمليات التثبيت وإلغاء التثبيت".

باتت الآن خدمتك جاهزة:

Services

File Action View Help

Services (Local)

MyService by Eng27

Start the service

Description:
MyService By Eng27

Name	Description	Status	Startup Type	Log
MessagingService_26dd6c0	Service sup...		Manual (Trig...	Loc
Microsoft (R) Diagnostics H...	Diagnostics ...		Manual	Loc
Microsoft Account Sign-in ...	Enables use...	Running	Manual (Trig...	Loc
Microsoft App-V Client	Manages A...		Disabled	Loc
Microsoft iSCSI Initiator Ser...	Manages In...		Manual	Loc
Microsoft Passport	Provides pr...		Manual (Trig...	Loc
Microsoft Passport Container	Manages lo...		Manual (Trig...	Loc
Microsoft Software Shadow...	Manages so...		Manual	Loc
Microsoft Storage Spaces S...	Host service...		Manual	Net
Microsoft Store Install Service	Provides inf...	Running	Manual	Loc
Microsoft Windows SMS Ro...	Routes mes...		Manual (Trig...	Loc
MSC.Licensing_11.13		Running	Automatic	Loc
MyService by Eng27	MyService B...		Manual	Loc
Natural Authentication	Signal aggr...		Manual (Trig...	Loc
Net.Tcp Port Sharing Service	Provides abi...		Disabled	Loc
NetBalancerService	Helper servi...	Running	Automatic	Loc
Netlogon	Maintains a ...		Manual	Loc
Network Connected Device...	Network Co...	Running	Manual (Trig...	Loc
Network Connection Broker	Brokers con...	Running	Manual (Trig...	Loc
Network Connections	Manages o...		Manual	Loc
Network Connectivity Assis...	Provides Dir...		Manual (Trig...	Loc

Extended / Standard

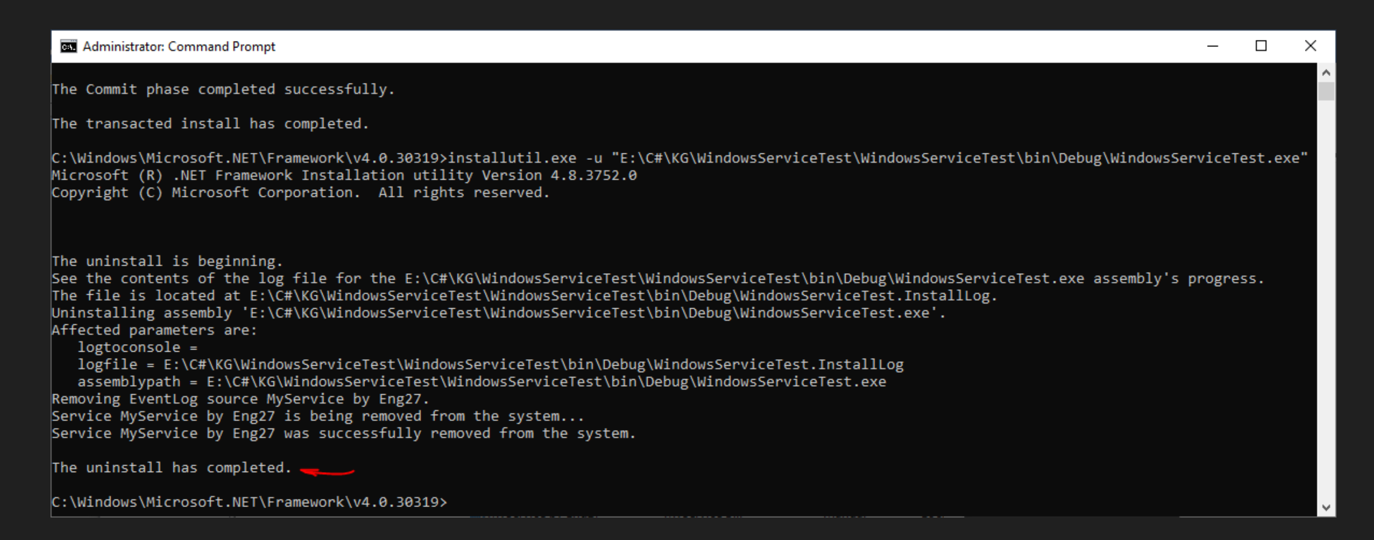
انقر على بدء Start، لبدء تشغيل الخدمة. عدل بعض الملفات ضمن القرص D (على اعتبار أن المجلد الذي تتم مراقبته هو القرص D بالكامل)، احذف ملفات أخرى، أعد تسمية غيرها، ثم انتقل لمجلد السجل Log الذي أنشأه الملف التنفيذي الممثل لخدمة ويندوز التي أنشأتها، ولاحظ التفاصيل.



ستعمل الخدمة تلقائيًّا عند تشغيل النظام.

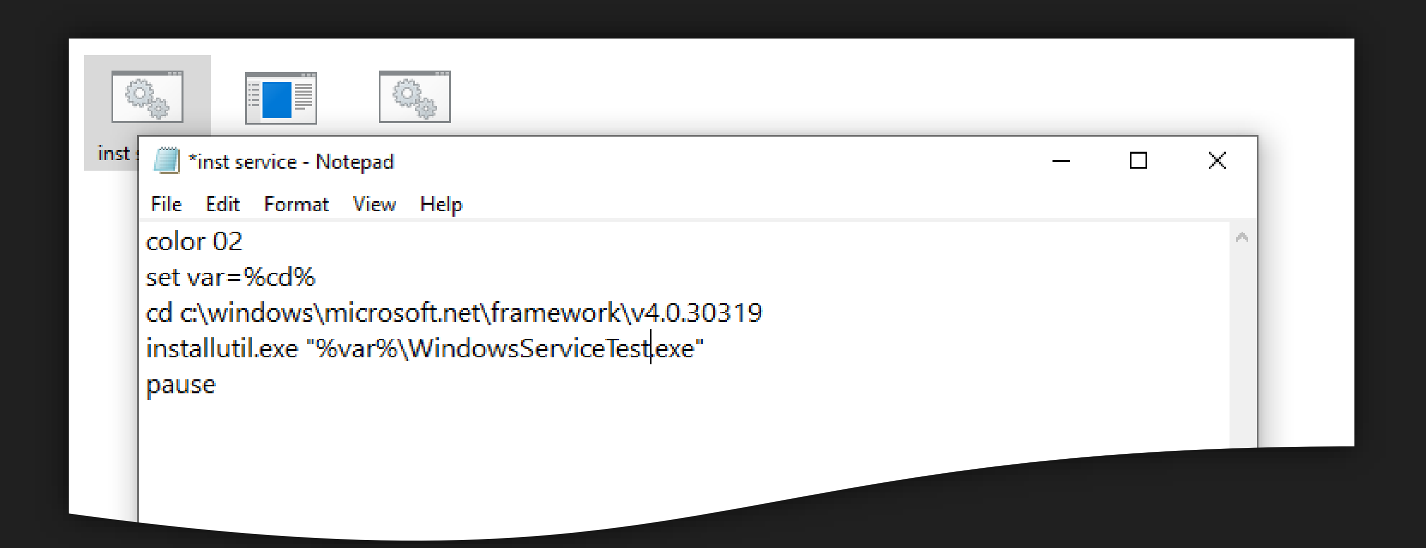
إلغاء تثبيت خدمة ويندوز

بدايةً، عليك إيقاف الخدمة أولًا قبل إلغاء تثبيتها، ثم استخدم نفس الكود الذي استخدمته لتثبيت الخدمة ولكن هذه المرة مع إضافة "–u" أمام الأمر installutil.exe:



أتمتة عملية التثبيت وإلغاء التثبيت

عوضًا عن تشغيل موجه الأوامر والانتقال لفهرس الأداة InstallUtil.exe ثم كتابة اسم الأداة وأمامها مسار الملف التنفيذي الممثل للخدمة، يمكنك تطوير ملف دفعي Batch File ليقوم بذلك عنك:



لكن لا تنس تشغيل الملف الدفعي كمسؤول.

وفي ختام هذه الفقرة (خدمات ويندوز)، أحيلك إلى موقع ميكروسوفت للمزيد حول خدمات ويندوز[1].

جهاز إدارة ويندوز WMI

جهاز إدارة ويندوز Windows Management Instrumentation هو تقنية تعطيك إمكانية الوصول لمعلومات عن نظام التشغيل، خدمات ويندوز، والعمليات الجارية في جهازك أو في جهاز متصل على الشبكة (بعد حصولك على الصلاحيات المطلوبة لوصولك لبياناته).

إذا كنت على اطلاع على موجه الأوامر والرجستري فإن كثيرًا من معلومات نظام التشغيل يمكنك الحصول عليها من خلالهما، بشكل مشابه فإن هذه التقنية تقوم بذلك لكن بشكل متقدم.

تطبيق 1 – لمحة سريعة

هذا التطبيق منقول حرفيًّا من موقع c-sharpcorner[2]، وهو كالتالي:

مقدمة[3]: يمكنك إنشاء نسَخ تجريبية من تطبيقاتك بمعرفة تفاصيل جهاز المستخدم مثل الـ ID الخاص بالكمبيوتر، القرص الصلب، المعالج، نظام التشغيل، أجهزة أخرى، وغيرها.

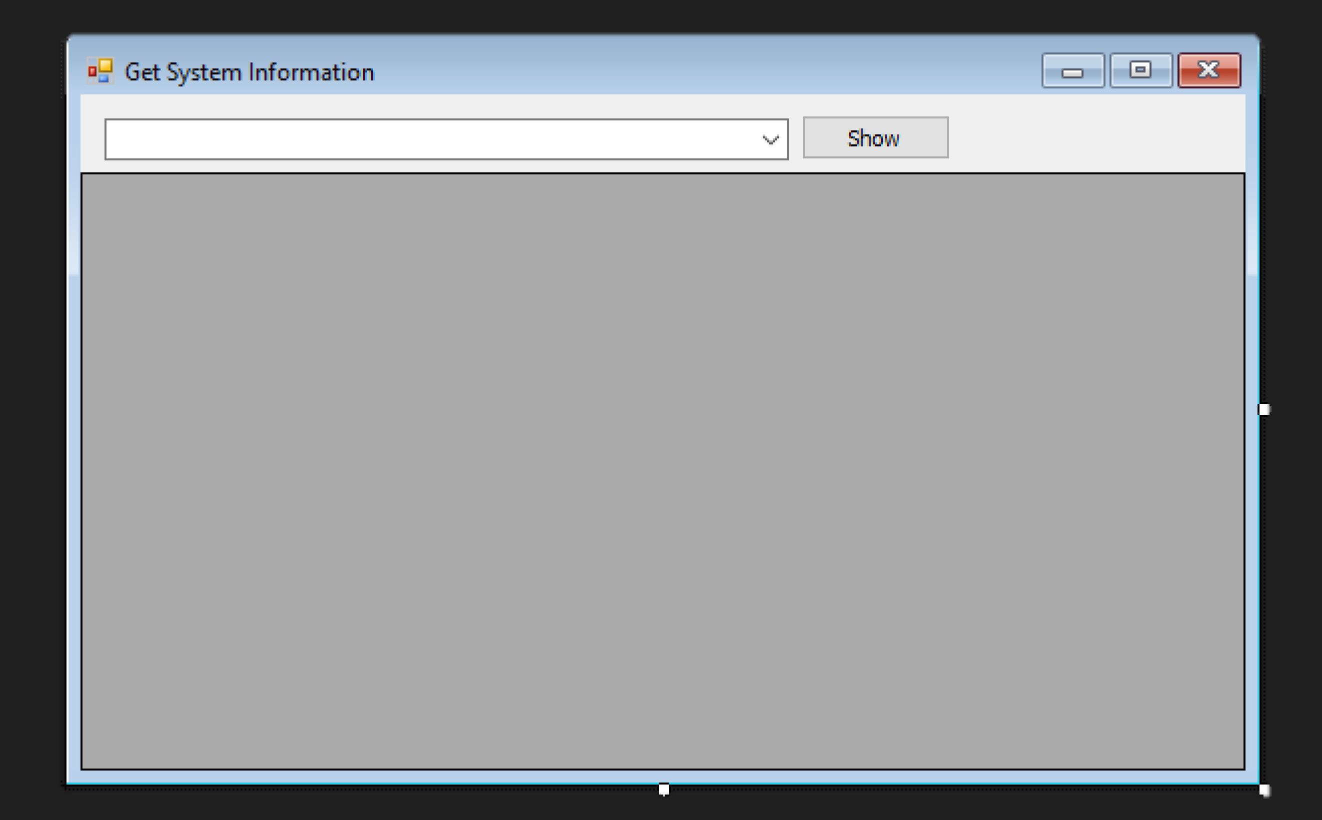
في البداية قم بإضافة المكتبة System.Management كمرجع من مراجع المشروع. ثم صمم الواجهة التالية:

---

[1] https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer
https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.eventlog.geteventlogs?view=dotnet-plat-ext-3.1

[2] راجع الرابط https://www.c-sharpcorner.com/uploadfile/75a48f/get-system-information-using-c-sharp-code/.

[3] المقدمة منقولة بتصرف.



(النافذة مصممة من قبل مؤلّف الكود)

استخدم الكود التالي:



```
using System;
using System.Collections;
using System.Management;
using System.Windows.Forms;

namespace GetSystemInformation
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnShow_Click(object sender, EventArgs e)
        {
            dgvWMI.DataSource = GetInformation(comboBoxWin32API.Text);
        }

        private ArrayList GetInformation(string qry)
        {
            ManagementObjectSearcher searcher;
            int i = 0;
            ArrayList arrayListInformationCollactor = new ArrayList();
```

```
            try
            {
                searcher = new ManagementObjectSearcher("SELECT * FROM " + qry);
                foreach (ManagementObject mo in searcher.Get())
                {
                    i++;
                    PropertyDataCollection searcherProperties = mo.Properties;
                    foreach (PropertyData sp in searcherProperties)
                    {
                        arrayListInformationCollactor.Add(sp);
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }
            return arrayListInformationCollactor;
        }
    }
}
```

يمكنك تحميل المشروع من الرابط المذكور في هامش بداية الفقرة.

عند تشغيل التطبيق – واختيار قسم معينة لإظهار بياناته – ستحصل على مجموعة من البيانات في أداة GridView، وهذا مجرد مثال لتفهم ما يمكنك الحصول عليه من خلال تقنية WMI، أما في الواقع فإنك ستبحث عن أمور معينة وتحفظها ضمن ملف معين لتتعامل معها لاحقًا لغرض أو لآخر.

فمثلًا في الكود السابق يمكنك بعد كتابة sp – وهو متغير مسؤول عن حفظ بيانات معينة – ثم نقطة، يمكنك الحصول على مجموعة من الخصائص التابعة للقسم المدروس (الذي اخترته ضمن القائمة المنسدلة).

تطبيق 2 – مكتبة جاهزة

هذا التطبيق منقول بتصرف عن موقع CodeProject [1]، وهو كما يلي:

يعطيك هذا التطبيق إمكانية الوصول لتفاصيل كثيرة في الحاسوب اعتمادًا على تقنية WMI، ويتميز عن التطبيق السابق بأنه أشمل ومفصَّل بشكل أوسع، وهو عبارة عن مشروع Console ومعه فئات كثيرة منظمة ضمن ثلاثة مجالات أسماء تمثل ثلاثة أقسام: الرجستري Registry والعمليات Process والعتاد المادي في الكمبيوتر Hardware. وقد جعلتها معًا على شكل مكتبة dll واحدة لأستخدمها على حاسوبي، مع الحفاظ على حقوق المؤلف.

حمل المشاريع الثلاثة من الرابط المذكور في هامش بداية الفقرة، وتابع معي:

بالنسبة لمشروع Registry، فإنه يمكنك من خلاله القيام بعمليات كثيرة مثل إنشاء المفاتيح والقيم والتعامل معها وأمور أخرى كثيرة، وقد ناقشنا الرجستري في كتابنا الأول.

أما بالنسبة لمشروع Process فيمكنك من خلاله معرفة العمليات الجارية، وإنشاء عملية جديدة، ومسح عملية سابقة، كما يلي:

- لإنشاء كائن يمكّنك من التعامل مع العمليات في ويندوز (لتتمكن من استخدام الأمثلة اللاحقة):



```
//using baileysoft.Wmi.Process; *must include
ProcessLocal processObject = new ProcessLocal();
```

المقصود بالتعليق المذكور أنه لو قمت بتحويل هذا المشروع وبقية المشاريع إلى ملف dll – كما فعلت أنا – فيجب عليك جعله مرجعا للمشروع والتصريح عنه بالكلمة using في بداية الكود.

[1] راجع الروابط:
https://www.codeproject.com/Articles/18122/Howto-Almost-Everything-in-WMI-via-C-Part-1-Regist
https://www.codeproject.com/Articles/18146/How-To-Almost-Everything-In-WMI-via-C-Part-2-Proce
https://www.codeproject.com/Articles/18268/How-To-Almost-Everything-In-WMI-via-C-Part-3-Hardw

- للحصول على العمليات الجارية:



```
Console.WriteLine("Fetching Running Processes: ");
foreach (string eachProcess in processObject.RunningProcesses())
{
    Console.WriteLine("Process: " + eachProcess);
}
Console.WriteLine("");
```

- لإنشاء عملية جديدة:



```
string processName = "notepad.exe";
Console.WriteLine("Creating Process: " + processName);
Console.WriteLine(processObject.CreateProcess(processName));
```

- لتغيير أولوية العملية:



```
Console.WriteLine("Setting Process Priority: Idle");
processObject.SetPriority(processName, ProcessPriority.priority.IDLE);
```

- لمعرفة مالك العملية:



```
Console.WriteLine("Process Owner: " + processObject.GetProcessOwner(processName));
```

- لمعرفة الـ ID الخاص بمالك العملية:



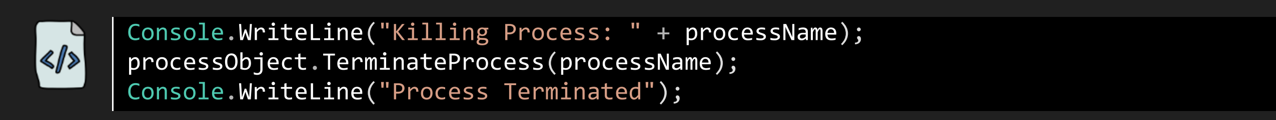
```
Console.WriteLine("Process Owner SID: " +
    processObject.GetProcessOwnerSID(processName));
Console.WriteLine("");
```

- لمعرفة تفاصيل كاملة عن عملية معينة:



```
Console.WriteLine("Properties of Process: " + processName);
foreach (string property in processObject.ProcessProperties(processName))
{
    Console.WriteLine(property);
}
Console.WriteLine("");
```

- لإنهاء عملية جارية:



```
Console.WriteLine("Killing Process: " + processName);
processObject.TerminateProcess(processName);
Console.WriteLine("Process Terminated");
```



لا تنسَ استخدام الكود الأول – تعريف العملية processName – مع جميع الأكواد.
يمكن استخدام أكثر من كود معًا.

وهذا مختصر مبدأ عمل أداة إدارة المهام Task Manager 😃، وبالمناسبة فإن الفئة System.Diagnostics.Process فيها إمكانيات كثيرة في هذا المجال، لكننا نناقش WMI وإمكانياتها لذلك لم نتطرق للفئة Process.

أما بالنسبة لمشروع Hardware فيمكنك من خلاله الحصول على تفاصيل كثيرة متعلقة بالعتاد المادي في الحاسوب. المشروع تم تقسيمه إلى فئات كما يلي:

1. Win32_BaseBoard: Mother board or System Board
2. Win32_Battery: System Battery
3. Win32_BIOS: System BIOS
4. Win32_Bus: Physical System Bus
5. Win32_CDROMDrive: System Optical Drives
6. Win32_DiskDrive: System Disks
7. Win32_DMAChannel: System DMA Channels
8. Win32_Fan: System Fan
9. Win32_FloppyController: System Floppy Controllers
10. Win32_FloppyDrive: System Floppy Drives
11. Win32_IDEController: System IDE Controllers
12. Win32_IRQResource: System IRQ Resources
13. Win32_Keyboard: System Keyboard

14. Win32_MemoryDevice: System Memory
15. Win32_NetworkAdapter: Network Adapters
16. Win32_NetworkAdapterConfiguration: Adapter configuration
17. Win32_OnBoardDevice: Common Devices built into the System board
18. Win32_ParallelPort: The Parallel ports
19. Win32_PCMCIAController: The PCMCIA Laptop bus
20. Win32_PhysicalMedia: Storage Media such as tapes, etc.
21. Win32_PhysicalMemory: The physical memory device
22. Win32_PortConnector: Physical ports such as DB-25 pin male, PS/2, etc.
23. Win32_Bus: Physical System Bus
24. Win32_PortResource: I/O ports on a system
25. Win32_POTSModem: Plain Old Telephone System Modem Devices
26. Win32_Processor: Processor specifications
27. Win32_SCSIController: System SCSI bus
28. Win32_SerialPorts: Serial Ports
29. Win32_SerialPortConfiguration: Port Configuration
30. Win32_SoundDevice: Sound Devices
31. Win32_SystemEnclosure: System Details
32. Win32_TapeDrive: Physical Tape Drives
33. Win32_TemperatureProbe: Heat Statistics
34. Win32_UninterruptiblePowerSupply: UPS details
35. Win32_USBController: USB Controller on a system
36. Win32_USBHub: USB Hub
37. Win32_VideoController: Physical Video Controller
38. Win32_VoltageProbe: Voltage Statistics

كل فئة من هذه الفئات مسؤولة عن جهاز معين، وهي جميعها مكوَّدة بنفس الصيغة، مع تغيير في العنصر المستدعى. بمعنى أن هذا المشروع ما هو إلا وسيط بينك وبين

مكتبات ميكروسوفت بحيث أنه يترجم أوامرك المكتوبة بلغة NET. (كما في أمثلة مشروع Process وكما سيتقدم من أمثلة لهذا المشروع) إلى أوامر أشبه بأوامر قواعد البيانات فيها ما يشبه هذا: select * from Win32_SerialPorts. أي أنك لو فهمت هذه الأوامر وكيفية التعامل معها لتمكنت من برمجة مشروع شامل مثل هذا، وربما حولته لمكتبة dll مثلما قمتُ مع هذا المشروع لتسهيل استخدامه. وللمزيد أُحيلك إلى موقع ميكروسوفت.[1]

- لإنشاء كائن يمكّنك من التعامل مع الأجهزة المتصلة بويندوز:



```
Connection wmiConnection = new Connection();
```

- للحصول على تفاصيل البطارية مثلًا:



```
Connection wmiConnection = new Connection();
Win32_Battery b = new Win32_Battery(wmiConnection);
foreach (string property in b.GetPropertyValues())
{
    Console.WriteLine(property);
}
```

بتغيير الفئة المُستنسَخة يمكنك الحصول على تفاصيل جميع الأجهزة المذكورة ضمن إمكانية هذا المشروع.

كما يمكنك التعامل مع الأجهزة المتصلة على الشبكة كما يلي:



```
//Requires UserName, Password, Domain, and Machine Name
Connection wmiConnection = new Connection("serverAccountUserName",
                                          "3l!t3p@$$",
                                          "BAILEYSOFT",
                                          "192.168.2.100");
```

[1] راجع الروابط:
https://docs.microsoft.com/en-gb/windows/win32/wmisdk/wmi-start-page?redirectedfrom=MSDN
https://docs.microsoft.com/en-us/previous-versions//aa394084(v=vs.85)?redirectedfrom=MSDN

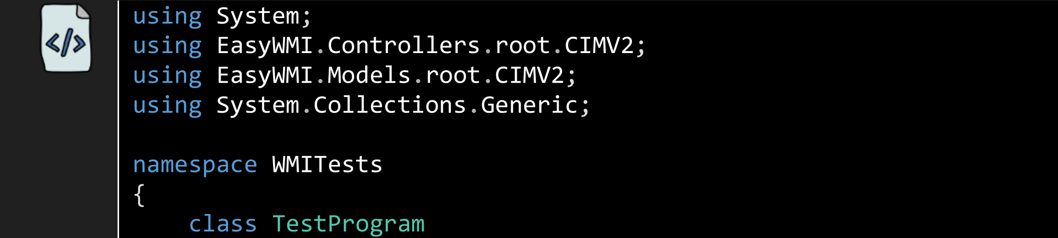
وبالمثل يتم استنساخ كائن من نوع ProcessRemote للعمليات وRegistryObject للرجستري.

تطبيق 3 – مكتبة جاهزة 2

على خطى التطبيق السابق فإن هذا التطبيق أيضًا يعطيك إمكانية الوصول لأقسام كثيرة في حاسوبك، لكن بشكل مختلف عن التطبيق السابق. لا يوجد اختلاف كمبدأ عن خطوات استخدام المكتبة – في حال تحويلها إلى ملف dll أو إبقاءها على حالها – ولكن ستتغير طريقة استخدامها، أي أنه في البداية ستُسند المكتبة إلى مراجع المشروع ثم سنستنسخ كائنًا يمثل الجزء من الحاسوب الذي نبحث عنه ثم نطبق كودًا معيّنًا كما أخبرنا مبرمجه.

في البداية حمّل المشروع[1]، وستجده – بعد تحميله – باسم EasyWMI وفيه مشروعين، أحدهما Console والآخر DLL. شغّل المشروع للاستئناس وتأمّل نتائجه، ستحصل على نتائج كثيرة مثل تفاصيل الحاسوب الرئيسية ونظام التشغيل والأقراص وغيرها، وهي مجرّد مثال لا أكثر. تأمّل الأكواد المستخدمة في التابع Main والتوابع الأخرى المحيطة بها، لا تغرّك المظاهر، فمبدؤها بسيط.

أنشئ مشروعَ Console وأسند إليه المكتبة EasyWMI كمرجع من مراجعه، ثم استخدم الكود التالي للحصول على تفاصيل البطارية مثلًا:



```
using System;
using EasyWMI.Controllers.root.CIMV2;
using EasyWMI.Models.root.CIMV2;
using System.Collections.Generic;

namespace WMITests
{
    class TestProgram
    {
        static void Main(string[] args)
        {
            string ComputerName = null; // اسم الكمبيوتر
            if (args.Length > 0)
                ComputerName = args[0];
            else
                ComputerName = "localhost";
```

[1] راجع هذا الرابط: https://www.codeproject.com/Articles/21971/WMI-Interface-for-NET.

```
            // كائنات تمثل البطارية
            IList<Win32_Battery> BATTERY = null;
            Battery battery = null;
            try
            {
                battery = new Battery(ComputerName);
                BATTERY = battery.SelectAll();
            }

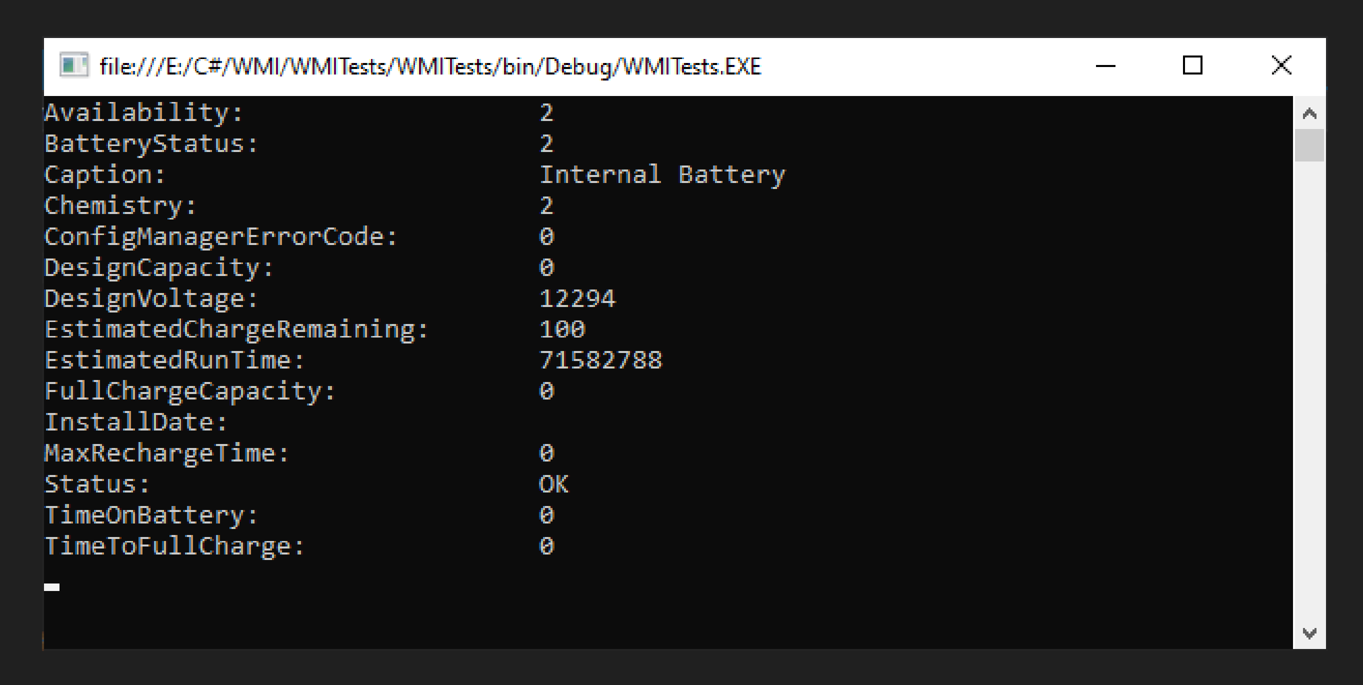
            catch (Exception ex)
            {
                Console.WriteLine("Error retrieving information.");
                Console.WriteLine("Message:  " + ex.Message);
            }

            Console.WriteLine("Availability:\t\t\t" +
                BATTERY[0].Availability);
            Console.WriteLine("BatteryStatus:\t\t\t" +
                BATTERY[0].BatteryStatus);
            Console.WriteLine("Caption:\t\t\t" +
                BATTERY[0].Caption);
            Console.WriteLine("Chemistry:\t\t\t" +
                BATTERY[0].Chemistry);
            Console.WriteLine("ConfigManagerErrorCode:\t\t" +
                BATTERY[0].ConfigManagerErrorCode);
            Console.WriteLine("DesignCapacity:\t\t\t" +
                BATTERY[0].DesignCapacity);
            Console.WriteLine("DesignVoltage:\t\t\t" +
                BATTERY[0].DesignVoltage);
            Console.WriteLine("EstimatedChargeRemaining:\t" +
                BATTERY[0].EstimatedChargeRemaining);
            Console.WriteLine("EstimatedRunTime:\t\t" +
                BATTERY[0].EstimatedRunTime);
            Console.WriteLine("FullChargeCapacity:\t\t" +
                BATTERY[0].FullChargeCapacity);
            Console.WriteLine("InstallDate:\t\t\t" +
                BATTERY[0].InstallDate);
            Console.WriteLine("MaxRechargeTime:\t\t" +
                BATTERY[0].MaxRechargeTime);
            Console.WriteLine("Status:\t\t\t\t" +
                BATTERY[0].Status);
            Console.WriteLine("TimeOnBattery:\t\t\t" +
                BATTERY[0].TimeOnBattery);
            Console.WriteLine("TimeToFullCharge:\t\t" +
                BATTERY[0].TimeToFullCharge);

            Console.ReadLine();
        }
    }
}
```

تعمّدت تسمية الكائنات بشكلين، أحرف كبيرة بالكامل وأخرى صغيرة بالكامل، يمكنك استخدام الأسلوب نفسه لسهولة التعامل مع الأجهزة الأخرى.

جرّب الكود، ستحصل على مجموعة من الأرقام التي لن تفهم دلالتها:



وفي الواقع فإن لهذه الأرقام دلالات معينة، وبالبحث عن Win32_Battery WMI ستحصل على صفحة من موقع ميكروسوفت [1] تشرح لك مجريات الأمور، وسأنقل بشيء من التصرّف بعض الخصائص لبيان الفكرة:

الخاصية Availability تعطيكَ وضعية الجهاز (على اعتبار البطارية جهاز من أجهزة الحاسوب)، فتأخذ القيمة 2 مثلًا إذا لم تكن وضعية البطارية معلومة، و3 إذا كانت تعمل أو كانت مليئة، و15 إذا كانت بوضع توفير الطاقة Power Save، وهكذا.

الخاصية ButteryStatus تعطيك حالة البطارية، فهي تأخذ القيمة 1 إذا لم تكن موصولة بالشاحن، و2 إذا كان النظام يعمل على طاقة التيار المتناوب AC والبطارية ليس بالضرورة أن تكون قيد الشحن، و3 إذا كانت مليئة، و4 إذا كانت منخفضة، وهكذا.

الخاصية Chemistry تعطيك التركيب الكيميائي للبطارية، فإذا كانت قيمتها 2 فهي غير معروفة، وما عداها معروف ومحدد بالنوع إلا القيمة 1 غير محددة بالنوع.

[1] راجع هذا الرابط: https://docs.microsoft.com/en-us/windows/win32/cimwin32prov/win32-battery.

الخاصية DesignCapacity تعطيك قيمة تخزين البطارية التصميمية مقدّرة بالـ mW/hr، في حال لم تكن الخاصية مدعومة يتم إرجاع القيمة 0 (مثل حالة المثال السابق).

الخاصية EstimatedChargeRemaining تعطيك نسبة الشحن المتبقية (الحالية).

الخاصية FullChargeCapacity تعطيك أكبر مقدار يمكن تخزينه ضمن البطارية من الطاقة مقدّرة بـ mW/hr. يمكن اعتبار أن عمر البطارية قد انتهى إذا انخفضت قيمة هذه الخاصية عن 80% من القيمة DesignCapacity. إذا لم تكن الخاصية مدعومة بالبطارية يتم إرجاع القيمة 0.

الخاصية MaxRechargeTime تعطيك الوقت الأعظمي مقدّرًا بالدقائق الذي تستهلكه البطارية لتُشحن بالكامل.

الخاصية TimeOnBattery تعطيك الوقت المنقضي منذ أن تم تبديل وضعية الطاقة إلى البطارية مقدّرًا بالثواني.

الخاصية TimeToFullyCharge تعطيك الوقت الأعظمي بالدقائق الذي تستغرقه البطارية لتُشحن بالكامل اعتمادًا على الاستخدام والتيار الحالي.

كما يمكنك الرجوع إلى الرابط المذكور لصفحة ميكروسوفت للحصول على تفاصيل أخرى عن البطارية وكيفية التعامل معها. وبالمثل يمكنك الحصول على بيانات لمختلف الأجهزة، فمثلًا، يمكنك الحصول على تفاصيل عن الأقراص كالتالي:



```
using System;
using EasyWMI.Controllers.root.CIMV2;
using EasyWMI.Models.root.CIMV2;
using System.Collections.Generic;

namespace WMITests
{
    class TestProgram
    {
        static void Main(string[] args)
        {
            string ComputerName = null;
            if (args.Length > 0)
                ComputerName = args[0];
            else
                ComputerName = "localhost";
```

```
            IList<Win32_LogicalDisk> LD = null;
            LogicalDisk ld = null;

            try
            {
                ld = new LogicalDisk(ComputerName);
                LD = ld.SelectAll();
            }

            catch (Exception ex)
            {
                Console.WriteLine("Error retrieving information.");
                Console.WriteLine("Message: " + ex.Message);
            }

            Console.WriteLine(LD.Count);

            Console.ReadLine();
        }
    }
}
```

في المثال الأخير عرضت فقط عدد الأقراص، وعلى أساسه سيتم التعامل مع الأقراص، مع اعتبار أن أول قرص دليله 0، وهكذا.

إمكانيات التعامل مع الأقراص كثيرة ولا يسعنا ذكرها هنا، ولا طائل من ذكرها بعد وضوح الفكرة من المثال ما قبل الأخير، وبالمثل يمكنك الحصول على تفاصيل أي جهاز ضمن الحاسوب المتوفر ضمن المكتبة الأخيرة، فقط عليك تغيير الكائنات المستنسَخة، وتغيير الخصائص التي ستحصل عليها من الأجهزة المراد التعامل معها، وعلى أساسه يمكنك برمجة برمجيات يمكنها التحكم بالحاسوب على أساس مواصفاته، بالإضافة إلى إمكانية إنشاء نسخ تجريبية من التطبيقات اعتمادًا على مواصفات أجهزة المستخدمين كما وضّحنا سابقًا.



يمكنك الحصول على بعض التفاصيل المفيدة من خلال الفئة Environment. كما يمكنك الاستفادة من الأداة PerformanceCounter كما تقدّم. هذا فضلًا عن استخدام موجّه الأوامر.

الفصل الرابع – البنية التحتية للغات البرمجة

قبل أن نبدأ في لب موضوع الكتاب، سنتناول بعض المواضيع الشائعة في عالم البرمجة، خصوصًا في صفوف المبتدئين. صحيح أن هذه المواضيع بديهية، ويفترض أن كثيرًا من الناس يعلمها، إلا أنها غامضة بالنسبة لكثير من الناس أيضًا، فكما يقول المثل: اللي ما بيعرف الكمبيوتر بقول عنه أتاري!

ما هي لغة البرمجة؟

بدايةً دعنا نسأل ماهي اللغة الإنسانية (أو البشرية)؟ ماهي اللغة العربية؟ ماهي اللغة الإنكليزية؟ ماهي اللغة أساسا؟

يمكن أن نقول – باختصار – أن اللغة هي وسيلة تخاطب، هي أداة تعطيك إمكانية التواصل مع غيرك. للغة نفسها أدوات تشكلها، ففيها بشكل أساسي الحروف، ومن الحروف تتشكل الكلمات، ومنها الجمل. موقع الكلمات ضمن الجملة يشكل معناها، ووجود كلمات من عدمه يغير تركيبها.

وبالعودة لحديثنا، لغة البرمجة هي وسيلة تخاطب بينك وبين المبرمجين بطريقة غير مباشرة، وبينك وبين الآلة بطريقة مباشرة، ولو أنه في الواقع هناك لغات وسيطة بين لغتك ولغة الآلة لكننا يمكننا اختزال جميع اللغات بين لغتك البرمجية ولغة الآلة بمفهوم واحد، وهو لغة البرمجة.

المكونات الأساسية للغات البرمجة ليست الحروف، كما في اللغات الإنسانية، وإنما الكلمات. ولكل كلمة صيغة تكتب على أساسها، ولتقريب الفكرة، الأمر أشبه بالجمل الإسمية مثلا في اللغة العربية (مبتدأ وخبر، حرف مشبه بالفعل واسمه وخبره، ...). بمعنى آخر: الجمل التي تكونها الكلمات البرمجية لها صيغ محددة، كما في اللغات الإنسانية.

تتميز اللغات البرمجية عن بعضها بوجود أو غياب مزايا معينة، تمامًا كما تتفاضل اللغات الإنسانية فيما بينها بوجود تراكيب أو غيابها. وهذا ما يجعل لغة البرمجة مناسبة لمجالات دون أخرى. للمزيد أحيلك لمنشور للباحثون المسلمون على صفحتهم على الفيسبوك من هنا [1].

ما هي أفضل لغة برمجة؟

كثيرًا ما تجد هذا السؤال على ألسنة المبتدئين أو ممن لم يعتنقوا لغةً برمجية بعد، وكأن البرمجة مسار واحد وشيء واحد وطريق واحد!

عليك أولًا تحديد المجال البرمجي الذي تود دخوله، والذي قد يتفرع لفروع عديدة، وبعدها يمكنك تحديد لغة البرمجة المناسبة لك، ومع هذا فـ "ما هي أفضل لغة برمجة؟" سؤال

[1] الباحثون المسلمون – أنواع لغات البرمجة:
https://www.facebook.com/The.Muslim.researchers/posts/1398105213684662/

غير دقيق حتى بعد تحديد المجال وتصفح اللغات المتاحة، وإنما قد ينفع أن تسأل "ما هي أكثر لغة برمجة مناسبة في المجال الفلاني؟".

يمكنك بالمقابل أن تقول "ما هي المجالات التي يمكنني أن أقوم بها من خلال لغة برمجة ما؟"، فكلما كانت لغتك البرمجية قادرة على مجالات أكثر كانت خبرتك ومعرفتك بهذه المجالات أكثر، فلا تحتاج في كل مجال تدخله أن تتعلم أساسيات اللغة ثم أساسيات المجال ثم تتوسع فيهما، بل يكفيك تعلم لغة البرمجة مرة واحدة والانتقال بين المجالات التي تخدّمها اللغة بسهولة ويسر.

من الأسئلة الدرامية أيضًا هي التي تقارن بين اللغات من المجال الواحد بشكل مطلق، فتجد أحدهم يسأل: أيها أفضل؟ لغة1 أم لغة2؟ أم لغة3؟

فلو كان السؤال أي اللغات أفضل في كذا لكان السؤال منطقيًّا، أما أن تبحث عن لغة أفضل من جميع اللغات في مجال ما فلن تجد ضالتك.

ماذا لو انقرضت لغتي البرمجية؟!

يكثر أيضًا هذا السؤال بين المبتدئين خصوصًا وقد تجده بين عامة المبرمجين، وكأن لغة البرمجة تريند Trend عليك لحاقه حتى لا تكون متخلّفًا أو قديمًا! فلغات البرمجة – وسائر تقنيات الحاسوب – ليست إلا أدوات أنت من تسيرها وليس العكس!

صحيح أن التقنيات الحديثة أفضل، إلا أن التقنيات القديمة ليست سيئة. وأنا لا أدعو الناس هنا لترك أحدث التقنيات والبحث عن القديم منها، بل على العكس، ولكن ما أقوله هو أن لغتك البرمجية – أو أي تقنية أخرى – مهما تقدم الزمن وحتى لو مات كل من يفهمونها، طالما أنت يمكنك استخدامها والاستفادة منها، فهي شيء مفيد ينبغي الاحتفاظ به والاعتماد عليه.

كيف يمكن إنشاء لغة برمجة؟

يعجبني فيك شغفك واهتمامك ورغبتك بإثراء وإغناء الثروات والإنجازات البرمجية العربية وأملك بعدم بقاء العرب توابع للغرب، لكن لا أريد إحباطك بالأسئلة التي سأطرحها عليك،

والتي تتمثل بـ 1) هل ستحمل لغتك مفاهيم جديدة؟؟ 2) هل لك علم بلغات البرمجة العربية الموجودة، والصعوبات التي واجهتها، والمزايا التي تتميز بها؟ 3) لماذا ترغب بوجود لغة برمجة تمثل العرب طالما لغات برمجة الغرب تعطيك كل ما تحتاج وزيادة، بما في ذلك شروحات عن كل شاردة وواردة في اللغة سواءً من الشركة المالكة للغة أو من أناس هواة أو مختصين لا علاقة لهم بالشركة؟

أما 1، فلا فائدة إطلاقا من إنشاء لغة ذات واجهة باللغة العربية، وهي أساسًا نسخة عن لغة أخرى، والأصح إن كنت ستفعل ذلك – أن تعرّب لغة برمجة ما – هو أن تسميها باسمها ولا تنسبها لنفسك. أما بالنسبة للمفاهيم الجديدة فأعني بها المتغيرات والحلقات والمصفوفات والكائنات وهياكل البيانات وما شابه ذلك، هذه كلها مفاهيم قد تتواجد في لغات وتغيب في أخرى، ما أحاول قوله هو أنني أرغب بأن أرى لمسات عربية حقيقية في لغتك لا تتواجد في لغات أخرى بشرط أن تكون هذه اللمسات ذات فائدة، مثلًا اخترع مفهوم الفاعل والمفعول به في لغتك 🤔 (بحيث يكون لهذا المفهوم غاية ما)، كـ if وelse والتي تقوم بالتحكم بأكوادك. فهمتني؟؟

وأما 2، فهناك لغات برمجة عربية كثيرة، مثل عمورية ولوغو عربي ولغة ج و"لغتي" والعنقاء ولغة كلمات ولغة زاي ولغة قلب ولغة إبداع ولغة س [1]، ولغة ألف. وأغلب هذه اللغات ليست إلا مشاريع. وبعضها لا زال قيد التطوير، وكلها ينقصها الدعم المادي والتقني. فمن غير المعقول أن تبدأ مشوارًا ابتدأه غيرك وتوقف لأنه اكتشف انسدادَ الطريق، فقط لأجل الحميّة!!

وبالنسبة لـ 3، طالما أن لغات البرمجة الغربية تنفعني وتؤدي أغراضي، فلا حاجة لي لابتكار لغات جديدة.

أما إذا كان الموضوع تعليميًّا، وللعلم بالشيء، ولاكتشاف المبادئ، فأنا معك قلبًا وقالبًا، ولي اهتمامات بهذا الموضوع، كاختراع لغة برمجة جديدة لأهداف تعليمية، طبعا لا تنس أن الكثير من لغات البرمجة ما كانت إلا مشاريع صغيرة حصلت على دعم من شركات

---

[1] راجع ويكيبيديا – قائمة لغات برمجة عربية.

كبيرة وأصبحت ذات شأن عالميًا. وبهذا الخصوص سأسرد معك ما تحتاجه لاختراع لغة برمجة (كما فهمت من هنا وهناك):

كبداية ضع نصب عينيك أنك تخترع لغة، واللغة هي وسيلة تواصل، وكما أن اللغات الإنسانية تعطي البشر إمكانية التواصل فيما بينهم، فلغات البرمجة تعطي البشر إمكانية التواصل مع الكمبيوتر. لاختراع أي لغة تحتاج إلى كلمات تسمى Reserved Words أو Keywords وإلى قواعد Syntax تربط هذه الكلمات مع بعضها لتشكل جملًا مفهومة وذات دلالات معينة. أو بكلام أدق فإنك تحتاج لرموز Tokens وقواعد Grammars.



لا أقصد بالرموز الإشارات مثل + و # و .. وإنما أقصد القيم الرمزية التي تحمل مكانة ما، ابحث عن الكلمة Token وستحصل على شيء يشبه هذا الشرح:

Something that represents or is a sign of sth else.

الرموز هي مجموعة من المحارف التي تملك معانٍ محددة (كالأحرف الأبجدية وعلامات الترقيم والأرقام و...)، ومجموعة من التراكيب اللغوية (كالفعل والفاعل، المبتدأ والخبر، الصفة والموصوف، إلخ)، كما يمكن أن يكون أكثر من رمز معًا ليشكلوا كلمات ذات دلالات محددة. فمثلًا في اللغة العربية لدينا "أ"، "؟"، "1" هي رموز تشكل محارف اللغة، و"أحمد" و"خالد" هي مجموعة من الرموز تشكل مع بعضها كلمات، كما أن "فوق" و"تحت" و"أمام" هي مجموعة من الرموز تشكل ما يسمى بالمفعول فيه.

أما القواعد فهي تربط الرموز أو مجموعة الرموز مع بعضها، فنجد أن جملة مثل "شربَ الطفلُ الحليبَ" لها شكل محدد هو {فعل، فاعل، مفعول به}، ولو تغير ترتيب مجموعات الرموز السابقة لتغير حكم الجملة (كـ "الطفل شرب الحليب" والتي أصبح الفاعل مبتدأً). أو مثلًا لدينا "الحديقة القديمة" هي {موصوف، صفة}، بينما "الحديقة قديمة" هي {مبتدأ، خبر}!!

كما يوجد في اللغات ما يسمى بالعبارات والجمل، فالعبارة هي مجموعة من الكلمات كالصفة والموصوف أو إن وأخواتها واسمها وخبرها أو ..، في حين أن الجمل هي مجموعة

من الكلمات والعبارات. ومثال على الجمل "أنا أعيش في منزل صغير"، والتي هي عبارة عن {مبتدأ، خبر (عبارة عن فعل)، تركيب أو عبارة} والتركيب بدوره يتكون من {حرف جر، موصوف (اسم مجرور)، صفة}. كما يمكنك اعتبار الجملة عبارة عن {فعل، وتتمة الجملة}.. أو غيرها من الاعتبارات، أي أن قواعد اللغة واسعة، وتختلف باختلاف المعاني التي يراد الوصول لها.

الآن بعد حصولك على الرموز Tokens والقواعد Grammars بإمكانك تشكيل أي لغة في العالم، سواءً أكانت لغة تواصل إنساني أم لغة برمجة.

في البرمجة لدينا الرموز التالية:

- رموز كلمات محجوزة Keywords Tokens: مثل else وdo وfor وreturn و...
- رموز حسابية Arithmetic Tokens: مثل =، =>، ==، ...
- رموز تعريفية (ترجمتها حرفيًّا) Identifier Tokens: *[a-zA-Z0-9_] [a-zA-Z_].
- رموز رقمية Number Tokens: *[0-9] [0-9]

لذا، إذا استطعت الربط بين الرموز السابقة فعندها يمكنك تشكيل لغة برمجة، كالجملة التالية: if (x == 5)، والتي هي من الشكل {Keyword (Identifier == Number)}. أي أن لغة البرمجة هذه هي عبارة عن *برنامج* يفهم كيف يربط الرموز المميزة للغة من خلال مجموعة من القواعد. وهو ما يسمى بالمترجم Compiler.

مع هذه المعلومات، وبعمليات بحث معدودة بإمكانك التوسع بالموضوع..

لكن لحظة.. لا تقلد!

قبل أن تشرع في اختراع لغة برمجة – عربية كانت أو أجنبية – لا تجعلها نسخة طبق الأصل عن غيرها، اللهم إلا أنك ترجمت الكلمات!

لا أحبذ رؤية كود بالشكل:

دالة مضروب (ع: صحيح): صحيح {

إذا ع <= 0 أرجع 1؛

ف: صحيح = ع؛

لكل ع = ع - 1، ع >= 2، --ع ف = * ع؛

أرجع ف؛

}؛ (لون الكلمات)

إلا إذا كان مشروعك تعليميًا أو تجريبيًا، وليس تجاريًا. فالأسطر السابقة ليست إلا تعريبًا ركيكًا للغة سي وأخواتها، ولو أنك إذا تمعنت فيها – أقصد الأسطر – لوجدتها منطقية وسليمة بلغة البرمجة سي شارب، ولكنك للحظة ستشعر أنك وضعت أحد أكواد اللغة على مترجم غوغل فحصلت على هذا الكود!

لا أريد إحباطك، ولا أريد إيهامك أيضًا، فالحقيقة المرة خير من الوهم الحلو، وربما لو كانت أول لغة لي (أول لغة أتعلمها وأكتب بها)، لو كانت مكتوبة بهذه الأسطر لاستغربت ممن يكتبون أكوادهم بالإنكليزية. واعلم أن للترجمة أصول، فلا يصح لمترجم أن يترجم لمجرد أنه يجيد اللغتين، وإلا لحصلت على كتب رديئة، تمامًا كأغلب كتبنا المترجمة، خصوصًا التي تسوّق لثقافة الغرب أو عاداتهم، أو لعلومهم، كأغلب كتب لغات البرمجة، المترجمة، أو بعض الكتب الجامعية. فعلى المترجم أن يلم باللغتين، وأن يجيد العلم الذي يترجمه باللغة التي يترجم منها، وأن يكون على اطلاع بالمصطلحات المتداولة باللغة التي يترجم إليها على ما يريد ترجمته.

وإني لا أدعي إلمامي بما أقول، فلا أنا ملم بالإنكليزية – ولا بأصول الترجمة بطبيعة الحال – ولست مطلعا بما يكفي على المراجع البرمجية العربية ولا حتى الإنكليزية؛ لذلك فكثيرًا ما ستجد مصطلحات قد تستغربها أو تجد أفضل منها، سواءً بالعربية أو بالإنكليزية.

وآمل من أصحاب الأسطر التي نقلتها قبل قليل ألا يظنوا أن كلامي ذمًّا أو تقليلًا من مجهودهم، فما أسسوا عليه اللغة أعمق وأكبر بكثير من الصيغة Syntax التي نناقشها في هذه الفقرة، ولكن ما أحاول قوله أنه من الممكن الوصول لتقنيات مماثلة أو مشابهة – أو قد تتفوق في مجالات معينة على – تقنيات الغرب (وفي حالتنا لغات البرمجة)، فإذا كان ما سننشئه ليس إلا نسخة معربة من منتجاتهم، فلْنستخدم منتجاتهم.

كيف يمكن احتراف لغة برمجة ما دون الأساسيات؟

لا يمكن [1] ☺️.

[1] في الواقع يمكن ذلك إذا كنت تعرف أساسيات لغة برمجة أخرى، فأنا انتقلت من VB لـ C# ولم أحتج الأساسيات (مع أنهما لغتان مختلفتان!!)، لكن غالبًا سيُفهم من السؤال احتراف لغة برمجة دون وجود خلفية برمجية إطلاقًا، لذلك قلت لا يمكن. بالمناسبة: السؤال يتكرر بشكل كبير.

الباب الثاني

الرسم والـ Graphics

أول خطوة عليك الإقدام عليها في طريقك للتصميم هي الرسم، وأعني بذلك رسم الأشكال البسيطة كالنقاط والخطوط والمضلعات والدوائر وغيرها، واختيار الألوان المناسبة. فمهما بدت الأدوات التي تستخدمها متقنةً ومعقدةً فهي بالآخر ذات شكل بسيط جرى عليها بعض التعديلات ورُسمت عليها بعض الأشكال لتظهر بالشكل الفاتن الذي تراه.

ولما كان كلّ شيء أمامك في الحاسوب عبارة عن رسومات، فعليك تقوية صلتك بكل ما يتعلق بالرسم لتصل لتصاميم متقنة.



الفصل الخامس – مدخل إلى الرسومات Graphics

يعتبر هذا الفصل أول الفصول الفعلية في هذا الكتاب، فالفصول الخمسة الأولى لم تكن إلا مقدمات سنبني عليها الفصول اللاحقة. يمكن اعتبار هذا الفصل أيضًا من فصول مقدمات الكتاب، فهو لا يحوي مواضيع متقدمة في الرسم، إلا أنه مع ذلك يضم بين فقراته كمّا لا بأس به من المعلومات.

أنصحك بقراءة هذا الفصل بتمعن وتطبيق أكواده وتطوير أمثلته، ثم العودة إليه مرة أخرى بعد انتهاءك من فصول تصميم الأدوات، فكثير من المعلومات التي ستقرؤها هنا لن تفهمها بشكل كامل إلا بعد الاطلاع على أمثلة تطبيقية تعتمد عليها.

مصطلحات ذات صلة

بعض المصطلحات التي سأوردها في هذه الفقرة قد تم ذكرها وبيانها سابقا، وبعضها قد لا تتعلق بمحتوى الفصل أصلًا، إلا أنني ارتأيت أن أجمعها معًا في فقرة واحدة لأنها متعلقة ببعضها، كما أنني سأضع آراء بعض المواقع في هذه المصطلحات أكثر من رأيي الشخصي بها.

ضع في ذهنك أن بعض المصطلحات في هذه الفقرة متعلقة بفصول لاحقة أكثر منها من هذا الفصل، ولكنها بالمقابل تؤسس لمصطلحات أخرى ضمن هذه الفقرة، لذلك لا تستغرب إذا اتجهنا خارج موضوع الرسم وما يتعلق به.

كما يمكنك أن تعتبر أن مصطلحات هذا الفصل هي مصطلحات عامة، في حين لو ذكرناها في الفصول التي تتعلق بها بشكل مباشر فإنها تصبح مصطلحات خاصة ذات توجهات معينة. فمثلًا يشير مصطلح UI بشكل عام إلى أي واجهة يمكن للمستخدم التعامل معها، أما لو استخدمناها مع الفصول التي تشرح تصميم الأدوات فإنها تصف العمليات التي يمكن أن تُجرى على هذه الأدوات للوصول لأفضل وأجمل شكل ممكن.

الـ Interface

الواجهة Interface هي عبارة عن تعريف لمجموعة من الوظائف، والتي يمكن لفئة أو تركيب Structure أن ترثها. وباستخدامك للواجهات فإنك تسمح لفئات برنامجك بالوراثة من أكثر من مصدر (فكما تعلم لا يمكن للفئات في #C الوراثة المتعددة). بالإضافة لما سبق فإنه يجب عليك الوراثة من الواجهات إذا أردت توريث التراكيب Structures، لأنها لا يمكنها الوراثة من الفئات أو التراكيب الأخرى. [1]

ويقال أيضًا أنها وصف للأفعال التي يمكن للكائن أن يقوم بها، فعلى سبيل المثال إذا حولت وضعية قابس المصباح فإنه يعمل (أو ينطفئ)، لا يهمك كيف حصل ذلك، ما يهمك أنه يقوم بوظيفته.. في OOP الواجهات هي وصف لجميع الوظائف التي يجب على

[1] ميكروسوفت – الواجهات (دليل برمجة #C)
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/

الكائنات القيام بها بغية الوصول لـ X، حيث X هي الغاية النهائية المطلوبة. مجددًا، أي شيء يعمل كالمصباح فإنه يجب أن يحوي على طرق مثل تشغيل_المصباح() وإيقاف_المصباح().. إذًا فالغاية من الواجهات هي ضمان وجود وظائف معينة في الكائنات تسمح بالوصول للغاية المرجوة. [1]

ويمكن اعتبار الواجهة عقد، كما في عالم البشر إذ تُلزِم العقودُ شخصين أو أكثر بالتصرف وفق محتويات هذه العقود، فالواجهات تحوي تصريحات وتعاريف لوظيفة أو عدة وظائف يمكن للكائنات تشاركها. العناصر التي ترث الواجهات يجب أن تُعرِّف جميع الوظائف المحتواة ضمن هذه الواجهات، بمعنى أن جميع أعضاء الواجهة يجب أن تكون معروفة بالنسبة لجميع ما سيرث من هذه الواجهة. في C# يمكن لفئة أو إجراء أن يرثوا من واجهة واحدة أو أكثر. [2]

وعلى ذمة المصدر الأخير أيضًا: يمكن للواجهات أن تضم خصائص أو طرق أو أحداث حتى، ويجب أن تكون جميع أعضاء الواجهات عامة Public، كما أنك لا تحتاج للتصريح عن أعضاء الواجهات بالكلمة public لأنها أصلًا كذلك. هذا بالإضافة إلى أن الواجهات يمكن أن تحوي فقط تصريحات ولا يمكنها ان تنفذ أكوادًا. والأهم من هذا كله فالواجهة يمكن لها أن ترث من واجهة أو أكثر!!

وقد ذكرنا سابقًا في الفصل صفر شرحًا يسيرًا عن مبدأ تعدد الواجهات في OOP، وألحقناه بكلام منقول من كتاب للأستاذ تركي العسيري، يمكنك العودة له لربط الأفكار مع بعضها.

وقد قال أحمد جمال خليفة في كتابه "فيجوال ستوديو 2008": الواجهة هي فئة مجردة abstract class [3] تحتوي على أعضاء وطرق مجردة، يمكن عمل نسخة منه بمفهوم Implementation [4] بدلًا من Inheritance، كما تتميز الوراثة من أكثر من واجهة على

---

[1] موقع cs.utah، بتصرّف – الواجهات https://www.cs.utah.edu/~germain/PPS/Topics/interfaces.html.
[2] موقع tutorialsteacher – الواجهات – C# https://www.tutorialsteacher.com/csharp/csharp-interface.
[3] الفئات المجردة لا يمكن استنساخها (لا يمكن إنشاء كائنات منها).
[4] الـ Implementation هي الوراثة من الواجهات، أما الوراثة من الفئات فتسمى Inheritance. عمومًا فإن هذا المصطلح أقرب ترجمة عربية له هي توظيف، أي أنك عندما ترث من واجهة ما فإنك توظّف أعضاءها ضمن الفئة التي جعلتها ترث منها. وما يميز الوراثة من الواجهات عن الوراثة من الفئات، أن بنود الواجهات عامة، تُخصَّص عند الوراثة، بينما بنود الفئات فهي محددة مسبقًا؛ لهذا فإن الوراثة من الواجهات يسمى توظيفًا لأعضاءها.

عكس الفئات التي لا ترث إلا من فئة واحدة. لذلك فاستخدام الواجهات هو البديل عن الوراثة المتعددة.

من الأمور المميزة في الواجهات هو أنه لا يمكن التصريح عن متغير عادي، ويُستعاض عنه بإنشاء خصائص، أي باستخدام مفهوم get وset. كما أنه لا يمكن كتابة أكواد داخل الواجهات ولا يوجد توابع بنّاءة في الواجهات لأنها مجردة أصلًا ولن يتم استنساخها!! [1]

الـ UI

واجهة المستخدم User Interface هي وصف لأي واجهة مرئية يمكن للمستخدم الوصول إليها. فيمكن لهذا المصطلح أن يشير إلى أنظمة التشغيل المرئية GUI والتي تتيح للمستخدم التفاعل مع الرسومات من خلال الحاسوب.

كما تُعرَف بأنها الواجهة التي تتيح لك التفاعل مع اللاعبين، والشخصيات، والكائنات الأخرى في الألعاب. أو أنها جهاز مادي مثل الطابعة والتي تحتوي على واجهة UI تعطي المستخدم إمكانية التفاعل مع خيارات الطباعة. [2]

ويمكن تعريفها بأنها كل شيء يمكن للمستخدم استخدامه للتفاعل مع الحاسوب، وهي ببساطة ما يسمح للمستخدم وأنظمة الحاسوب التفاعل مع بعضها البعض من خلال أجهزة الخرج والدخل. [3]

عادةً ما أُشبّه الـ UI بالمكياج، مع أنه كمصطلح يشير إلى شيء آخر، إلا أنه بجانب ما يحمله من معناه في هذا السياق فإنه يشير أيضًا إلى عمليات تصميم البرامج واختيار ألوان مكوناتها وأشكال خطوطها وحجومها وغيرها من الأمور التصميمية..

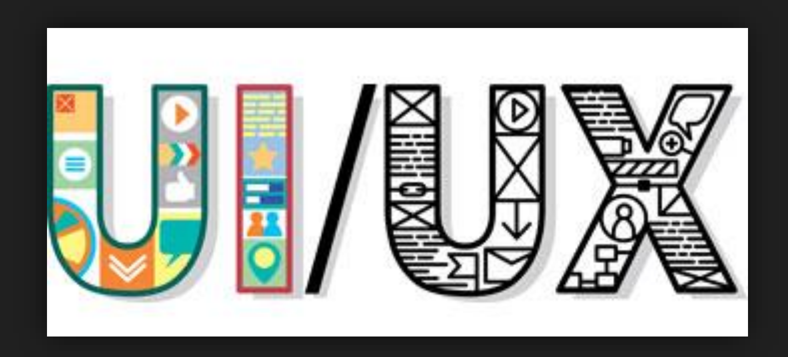
[1] انظر كتاب "فيجوال ستوديو 2008" لـ أحمد جمال خليفة (ص 180). وقد نقلت محتوى الفقرة التي تتحدث عن الواجهات حرفيًّا تقريبًا.
[2] موقع ComputerHope – ما هو الـ UI https://www.computerhope.com/jargon/u/ui.htm.
[3] موقع geeksforgeeks – الفرق بين CUI وGUI https://www.geeksforgeeks.org/what-is-the-difference-between-gui-and-cui/

الـ UX

تجربة المستخدم User Experience تصف ما يشعره المستخدم تجاه منتج ما، فالمنتجات سهلة الاستخدام والفهم بالنسبة للمستخدم تملك مستوى (أو مقدار أو تقييم) أعلى بالنسبة للـ UX، والعكس صحيح. [1]

هناك ما يسمى بالـ UX-Design، وهو ما يعرف بتصميم العمليات التي تستخدمها الفِرَق لإنشاء منتجات توفر تجارب مفيدة للمستخدمين. يتضمن ذلك تصميم عملية الحصول على المنتج بالكامل، بما في ذلك الأمور التي تتعلق بالعوامل التجارية والتصميم وسهولة الاستخدام والوظيفة. [2]

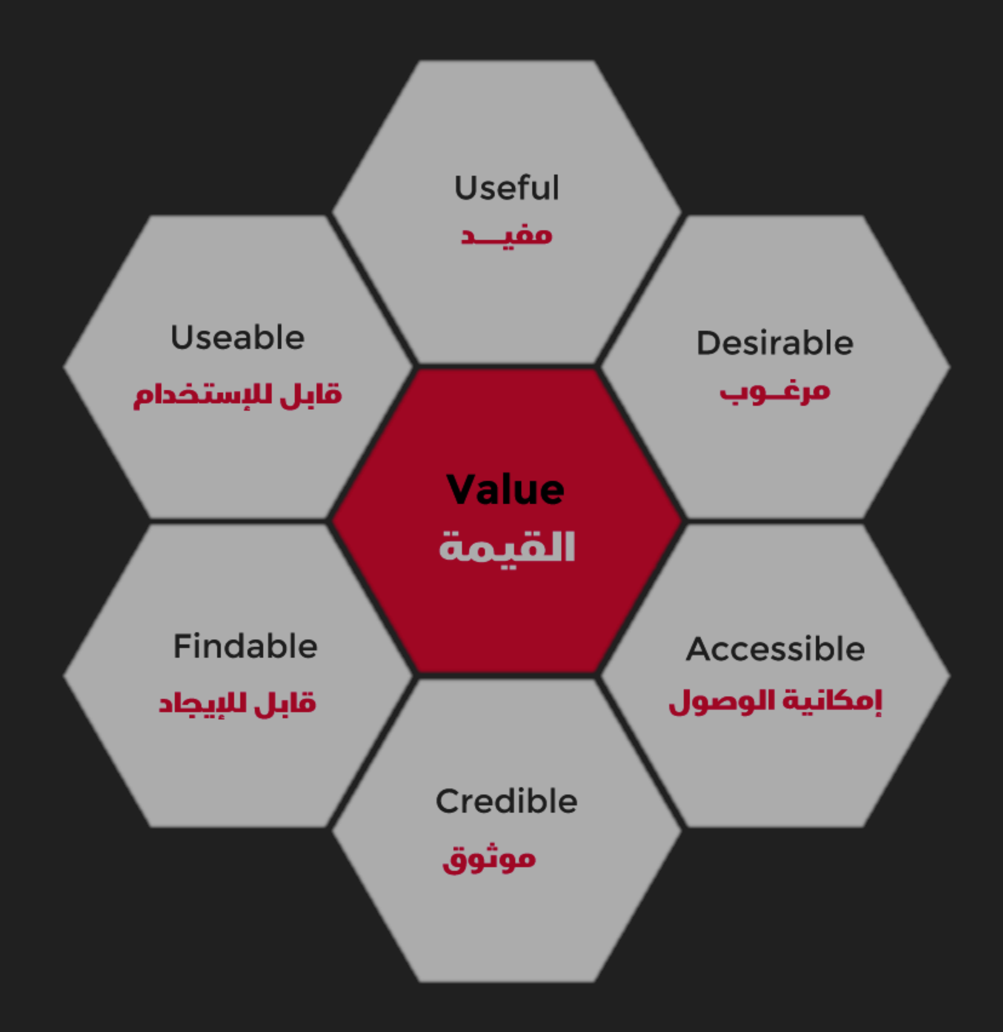
الصورة التالية تحمل في طياتها الفرق بين UI وUX (الصورة من موقع ComputerHope):



أما ويكيبيديا فقد قالت: هي كل ما يرتبط بسلوك وموقف وإحساس المستخدم حيال استخدامه منتجًا أو نظامًا أو خدمةً معينةً.

[1] موقع ComputerHope – ما هو الـ UX https://www.computerhope.com/jargon/u/ux.htm.
[2] موقع interaction-design، ما هو تصميم الـ UX https://www.interaction-design.org/literature/topics/ux-design.

يجب على التطبيقات أن تتضمن كل مما يلي لتحظى بتقدير تصميم UX جيد:



الـ CUI

واجهة المستخدم الرمزية Character User Interface هي واجهة مستخدم تتعامل مع الرموز، وفيها يتفاعل المستخدم مع الحاسوب باستخدام لوحة المفاتيح فقط. في هذا النوع من واجهات المستخدم للقيام بأي نشاط يجب إصدار أمر ما. [1]

الـ GUI

واجهة المستخدم الرسومية Graphic User Interface هي شكل من واجهات المستخدم التي تتيح للمستخدمين التفاعل مع الأجهزة الإلكترونية – كالحواسيب

[1] موقع geeksforgeeks – الفرق بين GUI وCUI
https://www.geeksforgeeks.org/what-is-the-difference-between-gui-and-cui/

والهواتف وغيرها – من خلال الأيقونات الرسومية والأشياء المرئية، عوضًا عن واجهات المستخدم التي تعتمد على النصوص (مثل البيوس BIOS)، أو الكتابة على موجهات الأوامر (مثل cmd وسطر أوامر Linux). [1]

كما تعرف بأنها واجهة مستخدم يتفاعل فيها المستخدم مع الحاسوب باستخدام الرسومات، حيث تتضمن الرسومات كلًّا من الأيقونات وأشرطة التنقل والصور وغيرها. يمكن استخدام الفأرة عند استخدام هذا النوع من الواجهات، وتُعتبَر واجهات صديقة للمستخدم User-Friendly ولا تتطلب أي خبرات. [2]

الـ GDI

واجهة رسومية متعلقة بأجهزة الحاسوب Graphics Device Interface هي خدمة API لبرامج ويندوز مسؤولة عن عرض كائنات رسومية ونقلها إلى أجهزة الخرج مثل الشاشات والطابعات. وهي مسؤولة عن مهام عديدة مثل رسم الخطوط والمنحنيات، والتعامل مع الخطوط والتحكم بقوالبها (يقصد بها قوالب الخطوط الجاهزة). هي ليست مسؤولة بشكل مباشر عن رسم النوافذ والقوائم وغيرها، حيث أن هذه المهام يتم تأمينها من خلال المكتبة user32.dll. [3]

وتُعرف أيضًا بأنها خدمة API تسمح للتطبيقات باستخدام الرسومات Graphics والنصوص المنسقة Formatted texts (قوالب الخطوط الجاهزة) على كل من أجهزة العرض (مثل الشاشات وأجهزة الإسقاط Projectors) والطابعات، حيث أن التطبيقات والبرامج لا يمكنها الوصول للأجهزة المسؤولة عن الرسومات (أجهزة الخرج) مباشرةً، وعوضًا عن ذلك فإن GDI تقوم بالتواسط بين الأجهزة المادية المسؤولة عن الرسومات في الحاسوب وبين التطبيقات. [4]

---

[1] ويكيبيديا، بتصرّف – واجهة المستخدم الرسومية https://en.wikipedia.org/wiki/Graphical_user_interface.
[2] موقع geeksforgeeks – الفرق بين GUI وCUI https://www.geeksforgeeks.org/what-is-the-difference-between-gui-and-cui/
[3] ويكيبيديا، بتصرّف – واجهة جهاز الرسومات https://en.wikipedia.org/wiki/Graphics_Device_Interface.
[4] ميكروسوفت – Windows GDI https://docs.microsoft.com/en-us/windows/win32/gdi/windows-gdi.



خدمات API ببساطة هي مجموعة من الأكواد البرمجية المكتوبة في ملفات معينة (مثل ملفات dll)، تؤدي وظائف معينة.
عند استخدامك إياها فإنك تستطيع القيام بأمور معينة بغض النظر عن قدرتك على تكويد هذه الأمور أم لا. فمثلًا يمكنك برمجة تطبيقات ويندوز وإضافة أدوات مثل الأزرار وصناديق النصوص وغيرها دون الحاجة لتكويد هذه الأشياء.

الـ +GDI

تعتبر +Windows GDI مثلها مثل GDI، إلا أن الأولى تدعم برامج Windows 64، وهي موجّهة لمنصة DotNET.[1] وعليه فإن نسخة الـ GDI لمنصة DotNET هي +GDI.[2]

برمجة الرسومات

تسمح لك +GDI كما أشرنا برسم الكائنات الرسومية المختلفة بما فيها النصوص والخطوط والمستطيلات والدوائر والأشكال الإهليلجية[3] والمضلعات وأشكال أخرى كثيرة.

مجالات أسماء +GDI

بمعرفة تفاصيل ووظيفة مجالات الأسماء يمكنك معرفة مجال الأسماء الذي يجب إضافته إلى مراجع المشروع لأداء أمور معينة.



يمكنك إضافة مرجع ما دون كتابته في قسم مراجع المشروع (باستخدام الكلمة using) بنقل المؤشر إلى كائنات الفئة المطلوب إسناد مرجعها والضغط على .+Ctrl.

[1] ميكروسوفت، بتصرّف – +GDI https://docs.microsoft.com/en-us/windows/win32/gdiplus/-gdiplus-gdi-start.
[2] موقع CodeProject، https://www.codeproject.com/Articles/4659/Introduction-to-GDI-in-NET.
[3] الشكل الإهليلجي: شكل منحني مغلق. يسمى أيضًا: قطع ناقص، أو شكل بيضوي.

الجدول التالي يوضّح مجالات الأسماء المتعلقة بالرسومات:

مجال الأسماء	الاستخدام
System.Drawing	فيه فئات عامة يمكن تطبيقها على مجالات الرسم، كما تدعم الألوان، والأقلام، والفُرَش[1]، والخطوط، وغيرها.
System.Drawing.Drawing2D	يوفر إمكانية التعامل مع الأشكال والمناطق المعقدة.
System.Drawing.Imaging	يوفر إمكانية التعامل مع الصور.
System.Drawing.Printing	يوفر إمكانية التعامل مع الطابعات، والحصول على معاينات الطباعة، والوصول لخصائص الطباعة.
Sustem.Drawing.Text	يعطيك إمكانية التعامل مع النصوص وتشكيلها.
System.Drawing.Design	يعطيك وظائف تدعم وقت التصميم Design-Time.



بعض المصطلحات المذكورة في الجدول السابق سيتم تفصيلها في فقرات لاحقة.

أدوات الرسم الأساسية

لتتمكن من الرسم في عالمنا البشري فإنك تحتاج لورقة – أو لوحة – للرسم عليها، وقلم لرسم الخطوط، وأدوات للتلوين. وبشكل مشابه فإنه للرسم في عالم البرمجة – بلغة C# – تحتاج كائنات من نوع Graphic تمثل مستويات الرسم، وكائنات من نوع Pen تمثل أقلام الرسم، تستخدم لرسم الخطوط والمنحنيات ومحيط الأشكال، كما تعطيك إمكانية تغيير ألوان الخطوط وسماكاتها وهل هي مستمرة أم متقطعة، وكائنات من نوع

[1] فُرَش: ج. فرشاة.

Brush تمثل فُرَش الرسم ومن خلاها يمكن ملء الأشكال المغلقة بالألوان أو بأشكال معينة، كما أنها تحدد كيف سيتم ملء النصوص عند التعامل معها كرسومات.

تعطيك كائنات الرسومات Graphics واجهات بين تطبيقك وجهاز العرض (شاشة الحاسوب)، وهذا ما تحدثنا عنه عند شرح الواجهات الرسومية.

يعطيك هذا النوع من الكائنات إمكانية الرسم من خلال مجموعة من الطرق، بعضها للرسم وبعضها للتلوين، ولهذا فإن الأولى اسمها يبدأ بـ Draw وتأخذ وسطاء من النوع Pen، والثانية يبدأ اسمها بـ Fill وتأخذ وسطاء من النوع Brush. مع استثناء بعض الطرق مثل DrawString والتي تأخذ Brush كوسيط.

الكائن من النوع Pen يحدد لون وعرض تنسيق الرسم.

كما تملك الكائنات الرسومية Graphics طريقتين لا غنى عنهما عند التعامل مع الرسومات هما Clear وDispose، حيث تعطيك الأولى إمكانية مسح السطح الرسومي بلون معين، وتضمن لك الثانية تحرير الذاكرة المستخدمة من قبل الكائنات الرسومية. لذلك فلزيادة فعالية تحرير الذاكرة المستخدمة قم باستدعاء الطريقة Dispose بعد الانتهاء من الرسم.

الجدول التالي يفصل بعض الطرق المستخدمة في الرسومات:

طرق الرسم Draw	طرق الإملاء Fill	الغاية من الطريقة وتفاصيلها
DrawArc	-	رسم قوس من قطع ناقص. تأخذ هذه الطريقة وسيطًا من النوع Rectangle لتحديد أبعاد القطْع ووسيطين لزاوية بدء الرسم وزاوية نهاية الرسم. كما يمكن إدخال أبعاد المستطيل مباشرةً كوسطاء للطريقة.

DrawBezier	-	رسم منحني بيزييه Bezier [1]. تأخذ أربع وسطاء تمثل نقاط المنحني. كما يمكنك إدخال إحداثيات النقاط مباشرة.
DrawClosedCurve	FillClosedCurve	رسم/إملاء منحني مغلق مار من مجموعة من النقاط من النوع Point. تأخذ هذه الطريقة مصفوفة من النقاط Point[ ] كوسطاء للإجراء. كما يمكن تحديد معامل الشد [2]، ولذلك يجب تحديد كيف سيتم ملء المنطقة المحدودة بالمنحني المغلق. مصفوفة النقاط يجب أن تحوي على الأقل أربع نقاط.
DrawCurve	-	رسم منحني، مار من مجموعة من النقاط من النوع Point. لا تختلف هذه الطريقة عن الطريقة السابقة من حيث التفاصيل (وسطاء الطريقة وأمور أخرى مثل معامل الشد) إلا أن مصفوفة النقاط يجب أن تحوي على الأقل ثلاث نقاط
DrawEllipse	FillEllipse	رسم/إملاء قطْع ناقص. ولها نفس تفاصيل الطريقة DrawArc إلا أنها لا تأخذ زوايا بدء أو انتهاء الرسم.
DrawIcon	-	رسم أيقونة من ملف معين.

[1] منحني بيزييه Bezier هو منحن طوره ونشره عالم الرياضيات بيير بيزييه Pierre Bézier (1910-1999) وهو مهندس فرنسي، وأحد مؤسسي أقسام النمذجة الفيزيائية والهندسية، وبالتالي أحد الذين طوّروا مجال التصميم والتصنيع الميكانيكي CAD/CAM، فمنحني بيزييه يستخدم في التصميم بمساعدة الحاسب (برامج التصميم الهندسية) وأنظمة الرسومات الحاسوبية. راجع هذا المقال https://en.wikipedia.org/wiki/Pierre_B%C3%A9zier.

[2] معامل شد المنحنيات هو قيمة أكبر أو تساوي الصفر وأصغر أو تساوي الواحد (إذا كانت أكبر من الواحد فستحصل على منحنيات غير مدروسة)، والذي يحدد تقوّس المنحني. كلما كان معامل الشد أصغر كان المنحني مشدودًا أكثر، بحيث يصبح المنحني مستقيمًا عندما يأخذ معامل الشد القيمة 0.

		وتأخذ وسيطًا من النوع Icon ووسيطين يمثلان إحداثيات هذه الأيقونة (الإحداثيات تمثل موقع الزاوية العليا اليسرا من الأيقونة). عند إنشاء كائن من النوع Icon عليك تمرير قيمة نصية تمثل مسار الأيقونة كوسيط للفئة Icon. هناك طريقة أخرى ترسم أيقونة دون تحجيمها (هذه الطريقة تغير حجم الأيقونة لتأخذ مساحة المستطيل الممر كوسيط).
DrawImage	-	رسم صورة من ملف معين. وتأخذ وسيطًا من النوع Image ووسيطين يمثلان موقع الصورة ضمن المنطقة المرسوم عليها كائن الرسومات Graphics. تعطيك هذه الطريقة عددًا كبيرًا من إمكانيات تمرير الوسطاء للحصول على نتائج مختلفة. هناك طريقتان أُخريان ترسمان الصور دون تحجيمها.
DrawLine	-	رسم قطعة مستقيمة. وتأخذ وسيطين من النوع Point تمثل بداية ونهاية القطعة المستقيمة، كما يمكن تمرير إحداثيات المستقيم مباشرةً.
DrawPath	FillPath	رسم/إملاء مسار. وتأخذ مسارًا من النوع GraphicsPath كوسط لها.
DrawPie	FillPie	رسم/إملاء مخطط دائري.

		وتأخذ هذه الطريقة كائنا من النوع Rectangle كوسيط يمثل حدود المخطط، ووسيطين يمثلان زاوية البدء وزاوية النهاية.
DrawPolygon	FillPolygon	رسم/إملاء مضلع. وتأخذ مصفوفة من النقاط [ ]Point تمثل زوايا المضلع.
DrawRectangle	FillRectangle	رسم/إملاء مستطيل. وتأخذ كائنًا من النوع Rectangle أو إحداثيات زاويته اليسرى وعرضه وارتفاعه.
DrawSting	-	رسم سلسلة. وتأخذ قيمة نصية sting كوسيط أول، ثم كائنًا من النوع Font، ثم كائنًا من النوع Brush، ثم كائنًا من النوع Point ليمثل الزاوية اليسرى من المنطقة الحاوية على السلسلة النصية. وكما أشرنا سابقًا فهذه الطريقة تأخذ وسيطًا من النوع Brush ولا تأخذ وسيطًا من النوع Pen.

والجدول التالي فيه بعض الطرق المفيدة في الرسم:

الطريقة	الغاية منها
BeginContainer	حفظ المحتوى الحالي لحاوية الرسم (محتوى الرسومات داخل منطقة الرسومات Graphics) وإرجاع قيمة من النوع GraphicsContainer، وفتح واستخدام حاوية رسومات جديدة.

Clear	مسح المحتوى الحالي للرسومات، وإملاء منطقة الرسومات بلون معين.
CopyFromScreen	نسخ المحتوى الحالي للنافذة ورسمه على منطقة الرسومات! تأخذ كائنًا من النوع Point يمثل موقع الزاوية العليا اليسرى من الشاشة، وكائنًا آخر من نفس النوع يمثل موقع الزاوية العليا اليسرى، وكائنًا من النوع Size يمثل حجم المنطقة المنسوخة.
Dispose	تحرير كل المصادر المستخدمة بالرسومات.
EndContainer	إغلاق حاوية الرسومات الحالية واستعادة حالة الرسومات لآخر حالة تم حفظها بالتابع BeginContainer. [1]
ExcludeClip	تغيير مساحة منطقة الرسومات.
IntersectClip	تغيير مساحة منطقة الرسومات إلى تقاطع منطقة الرسومات الحالية ومستطيل ما.
IsVisible	تحدد فيما إذا كانت نقطة ما معطاة بزوج من الإحداثيات موجودة ضمن المنطقة المرئية من مساحة منطقة الرسومات.
MeasureCharacterRanges	تأخذ مجموعة من الكائنات من نوع Region.
MeasureString	رسم سلسلة نصية مع ترك بعض الفراغات قبل وبعد القيمة النصية.
MultiplyTransform	تغيير جملة المحاور الإحداثية.
ResetClip	إعادة ضبط منطقة الرسومات.

[1] عند استدعاء الطريقة BeginContainer فإنك ستحفظ حالة الرسومات الموجودة قبل الاستدعاء في كائن من نوع GraphicsContainer، حيث أن حالة الرسومات تتضمن موقع المبدأ واتجاه المحاور ونسبة التكبير وغيرها. وعند تمرير هذا الكائن إلى الطريقة EndContainer فإنك تحذف حالة الرسومات الأخيرة وتستخدم حالة الرسومات المحفوظة ضمن الوسيط المُمَرَّر.
من الممكن أن تكون الحاويات متفرعة، وعليه، فإنه يمكنك استدعاء الطريقة الأولى عدة مرات ثم استدعاء الطريقة الثانية.

ResetTransform	إعادة ضبط جملة المحاور الإحداثية، وبذلك تصبح جملة المحاور الإحداثية بلا تحجيم Scaling ولا تدوير Rotation ولا نقل Translation.
Restore	استعادة حالة الرسومات لما يحتويه كائن من النوع GraphicsState.
RotateTransform	تطبيق دوران على جملة المحاور الإحداثية.
Save	حفظ حالة الرسومات الحالية وإرجاع قيمة من نوع GraphicsState.
ScaleTransform	تحجيم جملة المحاور الإحداثية.
SetClip	ضبط منطقة الرسومات على قيمة معينة.
TranslateClip	نقل منطقة الرسومات.
TranslateTransform	نقل جملة المحاور الإحداثية.



يمكن التصريح عن وسطاء الطرق السابقة قبل استدعاء هذه الطرق، أو أثناء ذلك. في الحالتين فإنك تمرر كائنات تمثل هذه الوسطاء.

هناك مجموعة من فُرَش الرسم الجاهزة كما في حالة الألوان Color، وذلك ضمن الفئة Brushes. وبذلك يمكنك استخدام أي فرشاة من الفُرَش الجاهزة عوضًا عن إنشاء واحدة.

لإنشاء فرشاة جديدة استنسخ الفئة SolidBrush.

كائنات الرسم الأساسية

كثيرة هي كائنات الرسم، ومن خلالها يمكنك إنشاء الرسومات. أهم كائنات الرسم التي يجب عليك معرفتها جيّدًا: النقطة Point، والحجم Size، والمستطيل Rectangle، والقلم Pen، والفرشاة Brush، وكائن الرسوم Graphics. وتأتي أهمية هذه الكائنات في أنها تمثل محتوى رسوماتك. وسنتناول هذه الكائنات في الفقرات التالية.

النقطة Point

تعتبر النقطة أهم كائنات الرسم على الإطلاق، فعلى أساسها سترسم أي شكل ترغب به، وعلى اعتبار أن رسوماتنا تكون على سطوح ثنائية الأبعاد فإن كائن النقطة له خاصيتان تعرفانه: الفواصل X والتراتيب Y.

الحجم Size

الحجم في البيئات الرسومية GUI هو مساحة الرسم، وليس مقدار ما يشغله الجسم من الفراغ كما في البيئات الفيزيائية (الكون الذي نعيشه). لكن، وعلى اعتبار أن مساحة الأشكال ثنائية الأبعاد هي مقدار ما يشغله الرسم من منطقة الرسم، فيمكن اعتبار أن مساحة الأشكال الرسومية هي حجمها. وعليه فإن حجم أي شكل رسومي – على اعتبار أن حجمه ومساحته سواء – يُعرَف بعرضه Width وارتفاعه Height.

المستطيل Rectangle

يشابه كائنُ المستطيل كائنَ النقطة بأهميته، فعليه يتم رسم الكثير من الأشكال. فمثلًا: في الـ GUI تُرسم الدائرة اعتمادًا على مستطيل يحيط بها (وكحالة خاصة للدائرة فإن المستطيل المحيط بالدائرة هو مربع). يتم التصريح عن كائن مستطيل بتمرير إحداثيات زاويته اليسرى العلوية وعرضه وطوله، أو بتمرير كائن نقطة يمثل زاويته العليا اليسرية وكائن حجم يمثل حجم المستطيل.

الأقلام Pen objects

هي كائنات تعطيك الإمكانية لرسم الخطوط والمنحنيات، والفئة Pen مجردة sealed، فلا يمكن الوراثة منها. يمكنك استنساخ قلم بتمرير وسيط من نوع فرشاة Brush أو لون Color، أو أحدهما مع عرض الخط (وقد يسمى سُمكه) [1].

على غرار كائنات الفرش، فهناك فئة اسمها Pens فيها أقلام جاهزة.

[1] إذا لم يتم تمرير عرض الخط سيتم اعتباره 1.

الجدول التالي فيه أهم خصائص كائنات الأقلام والقيم التي يمكن أن تأخذها:

الخاصية	قيمها	وظيفتها
DashCap	Flat Round Triangle	شكل نهاية أجزاء الخط
DashStyle	Custom Dash DashDot DashDotDot Dot Solid	شكل الخط
EndCap & StartCap	Flat Square Round Triangle NoAnchor SquareAnchor DiamondAnchor ArrowAnchor AnchorMask Custom	شكل نهاية الخط
LineJoin	Bevel Meter MiterClipped Round	شكل الوصلات بين الخطوط المختلفة (مثل الوصلات بين أضلاع المضلعات)

Width	قيمة عددية	سماكة الخط

فُرَش الرسم Brush objects

يمكنك إملاء المناطق المغلقة بألوان معينة، وذلك باستخدام فرش الرسم. كما يمكنك استخدام فرش الرسم الجاهزة وذلك من الفئة Brushes، وهي فئة مجرّدة.

وكما ذكرنا سابقًا فإن فرش الرسم تستخدم مع طرق إملاء الرسومات FillMethods بينما الأقلام تستخدم مع طرق رسم الرسومات DrawMethods.

ولفرش الرسم أنواع (وستراها في عدة أمثلة لاحقة):

- الفرشاة الصلبة SolidBrush: والتي تملأ الشكل بلون واحد.
- فرشاة التهشير [1] HatchBrush: والتي تملأ الشكل بلون مظلّل.
- الفرشاة المتدرجة LinearGradiantBrush: والتي تملأ الشكل بأكثر من لون بشكل متدرج.
- الفرشاة المتدرجة بشكل متشعب PathGradiantBrush: والتي تملأ الشكل بأكثر من لون، بحيث تبدأ بلون واحد وتنتهي بعدة ألوان.
- فرشاة الخامة TextureBrush: تملأ الشكل بصورة، ويتم تكرار الصورة حتى تملأ الشكل المطلوب ملؤه (كما تعمل الخاصية BackgroundImage عند ضبطها على القيمة Tile).

كائن الرسومات Graphics

كما ذكرنا سابقًا فإنك تحتاج سطحًا ترسم عليه، وهذا السطح هو الكائن Graphics، والذي يمكنك الحصول عليه من مصادر مختلفة.

تأتي أهمية كائن النقطة من أن الرسم يكون انطلاقًا منها، وبالمثل فإن أهمية كائن الرسومات من أنه عليه يتم الرسم.

[1] التهشير: ويسمى التظليل، وهو رسم خطوط متوازية أو متقاطعة في اتجاهات مختلفة.

كائنات أحداث الرسم PaintEventArgs

تزوّدك كائنات أحداث الرسم ببيانات مختلفة لأحداث الرسم، هذه البيانات تتضمن المنطقة الرسومية وما يمكن الرسم عليها. لتتعامل مع الرسومات عليك الوصول للخاصية Graphics التابعة لكائن مستنسخ من فئة أحداث الرسم (عادةً ما يكون باسم e)، وبعدها يمكنك القيام بكل ما تم سرده في فقرة أدوات الرسم الأساسية وحبة مسك!

وكبداية، سنستفتح الفصل بتطبيق جميل، وهو رسم مستطيل من خلال الفأرة.. لذلك فأنشئ مشروع نوافذ جديد واستخدم الكود التالي:



```
using System.Drawing;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        // متغيرات خاصة بالمشروع
        Rectangle rec;
        bool isDrawing = false;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_MouseDown(object sender, MouseEventArgs e)
        {
            // نقطة بدء الرسم، وهي زاوية المستطيل
            rec.X = e.X;
            rec.Y = e.Y;
            isDrawing = true;
        }

        private void Form1_MouseUp(object sender, MouseEventArgs e)
        {
            isDrawing = false; // إذا تم رفع النقر عن مؤشر الفأرة فسيتوقف الرسم عندها
        }

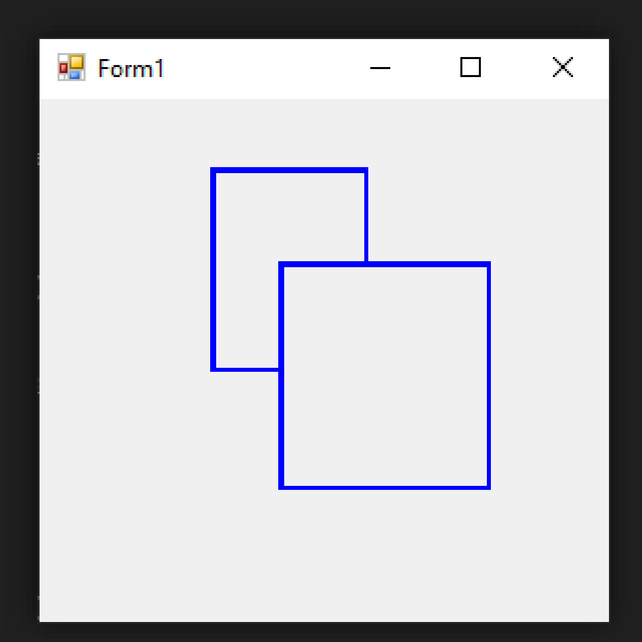
        private void Form1_MouseMove(object sender, MouseEventArgs e)
        {
            if (isDrawing)
            {
                if (e.X < rec.X)
                {
                    rec.Width = rec.X - e.X;
                    rec.X = e.X;
                }
                else
                    rec.Width = e.X - rec.X;
```

```
                if (e.Y < rec.Y)
                {
                    rec.Height = rec.Y - e.Y;
                    rec.Y = e.Y;
                }
                else
                    rec.Height = e.Y - rec.Y;

                this.Invalidate(rec); // يحفّز هذا الإجراء حدث الرسم
            }

            private void Form1_Paint(object sender, PaintEventArgs e)
            {
                // رسم مستطيل
                e.Graphics.DrawRectangle(new Pen(Color.Blue, 5), rec);
            }
        }
}
```

شغّل التطبيق، واسحب مؤشر الفأرة (من الأعلى للأسفل) على النافذة مع الضغط باستمرار لرسم مستطيلين، لتحصل على:



قد تبدو للوهلة الأولى العملية بسيطة، وهي كذلك فعلًا، لكنك قد تغير نظرتك إذا حاولت تغيير حجم النافذة، أو الرسم بعكس اتجاه الرسم السابق (سحب مؤشر الفأرة أثناء الضغط من الأسفل للأعلى). لا بأس، فتراكم المعلومات التي ستكتسبها خلال فقرات هذا الفصل ستجيب بعضًا من تساؤلاتك، فاستمر!

الرسم دون الاعتماد على أحداث الرسم

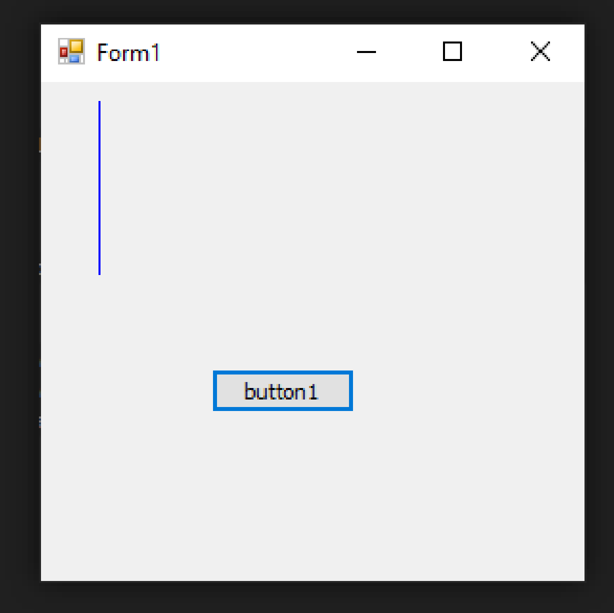
من المحتمل أن المثال السابق قد أربكك، ومع ذلك فلا تقلق فالأمثلة القادمة ستوضح الكثير (مع أني اعتمدت على النتائج التي سيحصل عليها القارئ بتطبيقه للأكواد وليس على قراءته لها من صفحات الكتاب).

لستَ مضطرًّا للتعامل مع كائنات أحداث الرسم (التي تحصل عليها من أحداث مثل Paint)، يمكنك الحصول على كائن الرسومات Graphics من كل الأدوات التي تملك الطريقة CreateGraphics، لاحظ:

```
using System.Drawing;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); }

        private void button1_Click(object sender, System.EventArgs e)
        {
            Point p1 = new Point(30, 10);
            Point p2 = new Point(30, 100);
            this.CreateGraphics().DrawLine(Pens.Blue, p1, p2);
        }
    }
}
```

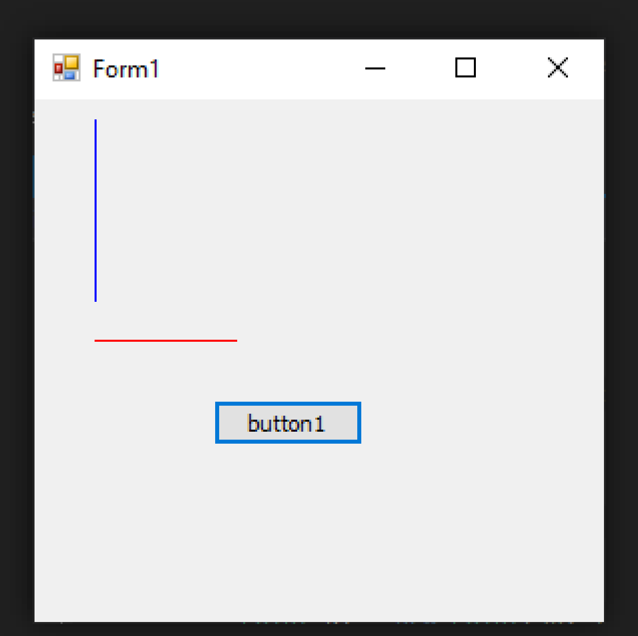


```
using System.Drawing;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, System.EventArgs e)
        {
            Point p1 = new Point(30, 10);
            Point p2 = new Point(30, 100);
            this.CreateGraphics().DrawLine(Pens.Blue, p1, p2);

            Point p3 = new Point(30, 120);
            Point p4 = new Point(100, 120);
            this.CreateGraphics().DrawLine(Pens.Red, p3, p4);
        }
    }
}
```



لكن بالمقابل، عند الرسم باستخدام الطريقة CreateGraphics لن تتمكن من تحديث الرسم، لذلك فاستخدام كائنات أحداث الرسم أولى.

دقة الرسم

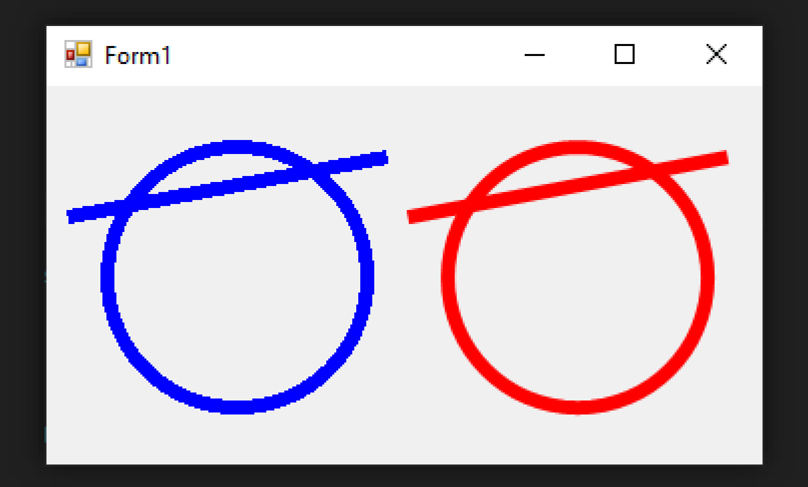
يمكن تحديد دقة الكائن الرسومي من خلال الخاصية SmoothingMode، والتي تأخذ قيمة معدد من نفس النوع.



```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            Pen p1 = new Pen(Color.Blue, 7);
            Pen p2 = new Pen(Color.Red, 7);
            g.DrawEllipse(p1, 30, 30, 130, 130);
            g.DrawLine(p1, 170, 35, 10, 65);
            g.SmoothingMode = SmoothingMode.AntiAlias;
            g.DrawEllipse(p2, 200, 30, 130, 130);
            g.DrawLine(p2, 340, 35, 180, 65);
            p1.Dispose();
            p2.Dispose();
        }
    }
}
```



كائن اللون

يمكنك التصريح عن لون جديد باستدعاء الطريقة FromArbg، إذ يمكنك تعريف ما يزيد عن 16 مليون لون (وهذا ما يفسر عندما تدخل إحداهن أحد محلات الألبسة وتتصفح ما يزيد عن 200 لون من تدريجات اللون الأحمر وتخرج دون اختيار أي قطعة بحجة "ما لاقيت اللون الي ببالي" 🤒 🤷🏻‍♀️ 💁🏼‍♀️ 🐍). كما يعطيك التركيب Color عشرات الألوان (نفسها الألوان التي تضبط الخصائص ForeColor وBackColor عليها).

تأخذ الطريقة FromArbg ثلاثة وسطاء لتمثيل نسب الألوان الأساسية الثلاثة: الأحمر والأزرق والأخضر على الترتيب، وذلك من القيمة 0 وحتى 255. كما يمكن أن تأخذ 4 وسطاء لتمثيل معامل الشفافية بالإضافة إلى الألوان الثلاث السابقة نفسها. معامل الشفافية يأخذ القيم ما بين 0 (عندها يكون اللون شفّافًا) و 255 (عندها يكون اللون معتمًا بشكل تام).

يمكنك الحصول على معكوس أي لون بطرح كل مكوّن من مكوناته الثلاثة من 255، لذلك فاللون الأبيض معاكس للون الأسود مثلًا. كما يمكنك الحصول على لون رمادي بجعل مكوناته الثلاثة متساوية، فكلما كانت مكوناته أقرب للـ 0 كان رماديّا غامقًا، والعكس بالعكس.

يمكنك تحويل الألوان لأعداد صحيحة لتتعامل معها لاحقًا إن احتجت ذلك، فنقل الأعداد ضمن ملفات (أو بين النماذج Forms والبرامج المختلفة) أسهل من نقل الألوان.

رسم بعض الخطوط

يمكن أن ترسم الخطوط بأنماط Styles متعددة، المثال التالي يعرضها في رسم واحد:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); }
```

```
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            Pen p = new Pen(Color.Black, 3);

            p.DashStyle = DashStyle.Solid;
            g.DrawLine(p, 5, 10, 205, 10);

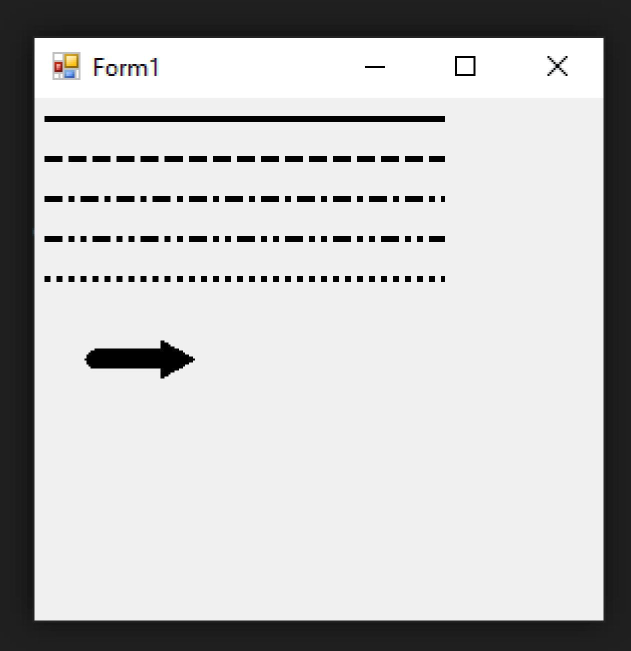
            p.DashStyle = DashStyle.Dash;
            g.DrawLine(p, 5, 30, 205, 30);

            p.DashStyle = DashStyle.DashDot;
            g.DrawLine(p, 5, 50, 205, 50);

            p.DashStyle = DashStyle.DashDotDot;
            g.DrawLine(p, 5, 70, 205, 70);

            p.DashStyle = DashStyle.Dot;
            g.DrawLine(p, 5, 90, 205, 90);


            p = new Pen(Color.Black, 10);
            p.StartCap = LineCap.Round;
            p.EndCap = LineCap.ArrowAnchor;
            g.DrawLine(p, 30, 130, 80, 130);
            p.Dispose();
        }
    }
}
```



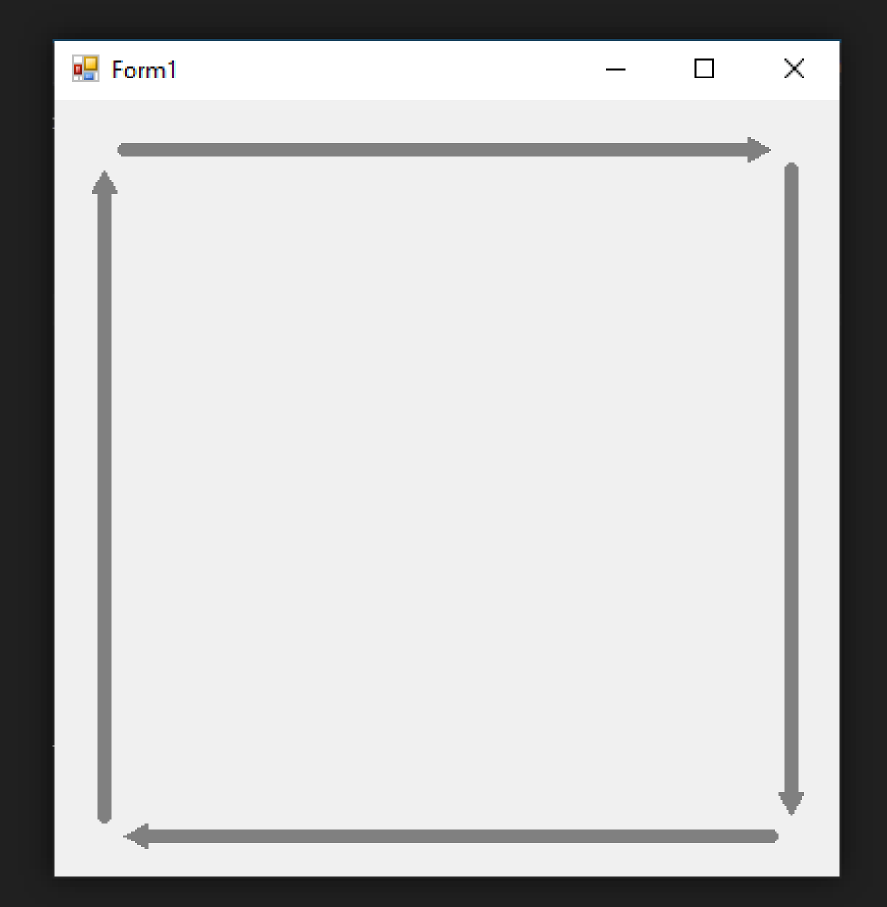
لاحظ هذا:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            Pen p2 = new Pen(Color.Gray, 7);
            p2.EndCap = LineCap.Round;
            p2.StartCap = LineCap.ArrowAnchor;
            g.DrawLine(p2, 25, 35, 25, 365);
            g.DrawLine(p2, 35, 375, 365, 375);
            g.DrawLine(p2, 375, 365, 375, 35);
            g.DrawLine(p2, 365, 25, 35, 25);
        }
    }
}
```



رسم بعض الأشكال وإملاءها

لرسم الحواف اعتمد على الطرق التي تبدأ بـ Draw، أما لإملاءها (لطلاءها بلون ما) فاستخدم تلك التي تبدأ بـ Fill. كما أنه إذا أردت رسم الأشكال المغلقة فاعتمد على كائنات من النوع GraphicsPath (سنتناول هذا النوع من الكائنات بشيء من التفصيل لاحقًا). سأرسم في المثال التالي دائرة ومربع ومثلث:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

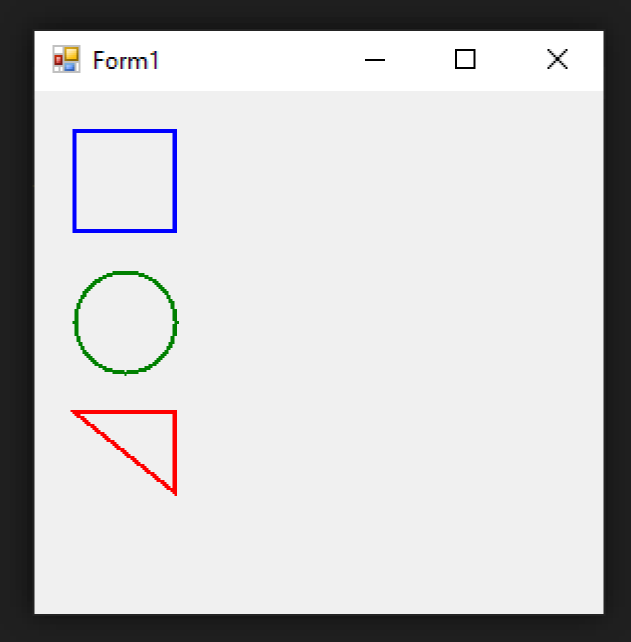
namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            Pen penBlue = new Pen(Color.Blue, 2.0f);
            Pen penRed = new Pen(Color.Red, 2.0f);
            Pen penGreen = new Pen(Color.Green, 2.0f);

            Rectangle r1 = new Rectangle(20,20, 50, 50);
            Rectangle r2 = new Rectangle(20, 90, 50, 50);

            GraphicsPath path = new GraphicsPath();
            path.AddLine(20, 160, 70, 160);
            path.AddLine(70, 160, 70, 200);
            path.AddLine(70, 200, 20, 160);
            path.CloseFigure();

            g.DrawRectangle(penBlue, r1);
            g.DrawEllipse(penGreen, r2);
            g.DrawPath(penRed, path);
        }
    }
}
```



أما إذا قمت بملء الأشكال عوضًا عن رسمها:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

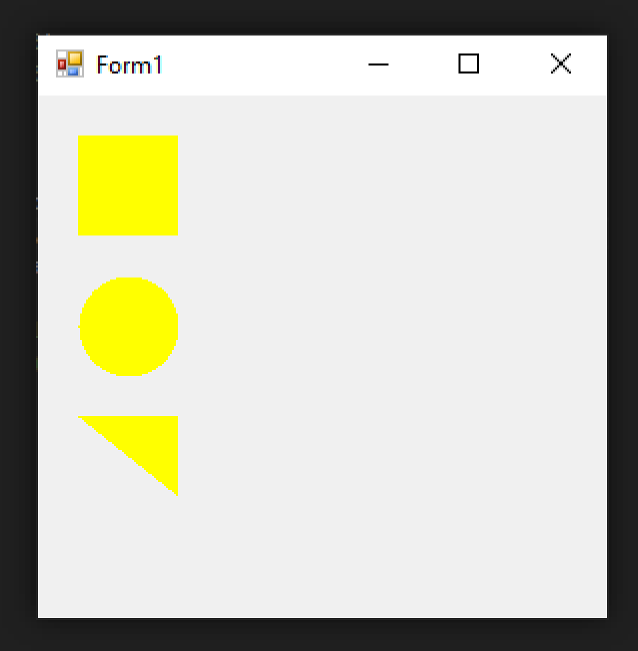
namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            Rectangle r1 = new Rectangle(20,20, 50, 50);
            Rectangle r2 = new Rectangle(20, 90, 50, 50);

            GraphicsPath path = new GraphicsPath();
            path.AddLine(20, 160, 70, 160);
            path.AddLine(70, 160, 70, 200);
            path.AddLine(70, 200, 20, 160);
            path.CloseFigure();

            g.FillRectangle(Brushes.Yellow, r1);
            g.FillEllipse(Brushes.Yellow, r2);
            g.FillPath(Brushes.Yellow, path);
        }
    }
}
```



فإذا أردت ملء الأشكال ورسمها في الوقت نفسه:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

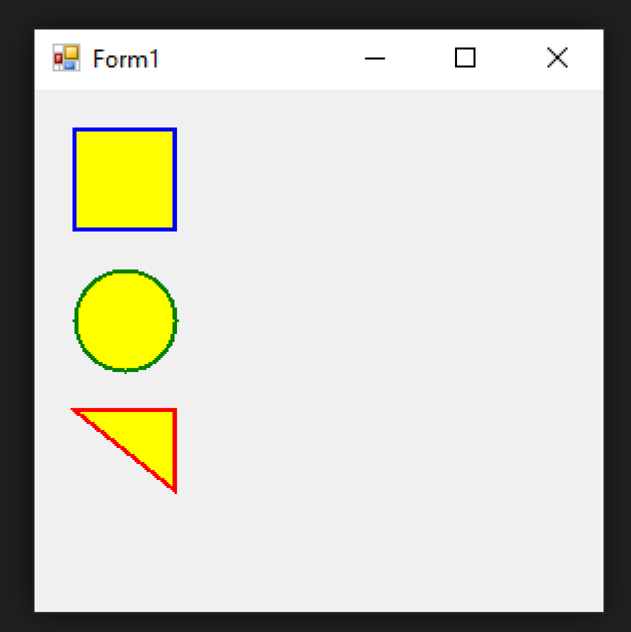
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            Pen penBlue = new Pen(Color.Blue, 2.0f);
            Pen penRed = new Pen(Color.Red, 2.0f);
            Pen penGreen = new Pen(Color.Green, 2.0f);

            Rectangle r1 = new Rectangle(20,20, 50, 50);
            Rectangle r2 = new Rectangle(20, 90, 50, 50);

            GraphicsPath path = new GraphicsPath();
            path.AddLine(20, 160, 70, 160);
            path.AddLine(70, 160, 70, 200);
            path.AddLine(70, 200, 20, 160);
            path.CloseFigure();

            g.FillRectangle(Brushes.Yellow, r1);
            g.FillEllipse(Brushes.Yellow, r2);
            g.FillPath(Brushes.Yellow, path);
```

```
            g.DrawRectangle(penBlue, r1);
            g.DrawEllipse(penGreen, r2);
            g.DrawPath(penRed, path);
        }
    }
}
```



هل لاحظت ما حدث؟ البرنامج لا يملك الوعي الكافي ليكتشف أن الخطوط الملونة بالأحمر والأخضر والأزرق هي حواف للأشكال الملونة بالأصفر، وإن وجود الحواف بمكانها الصحيح المتناسب مع الأشكال هو محض صدفة لا أكثر (بالنسبة للبرنامج).

لاحظ أننا رسمنا الأشكال أولًا ثم الحواف، وإلا فستكون الحواف ضعيفة بسبب أثر الأشكال الملونة عليها.

رسم نجمة



```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); }
```

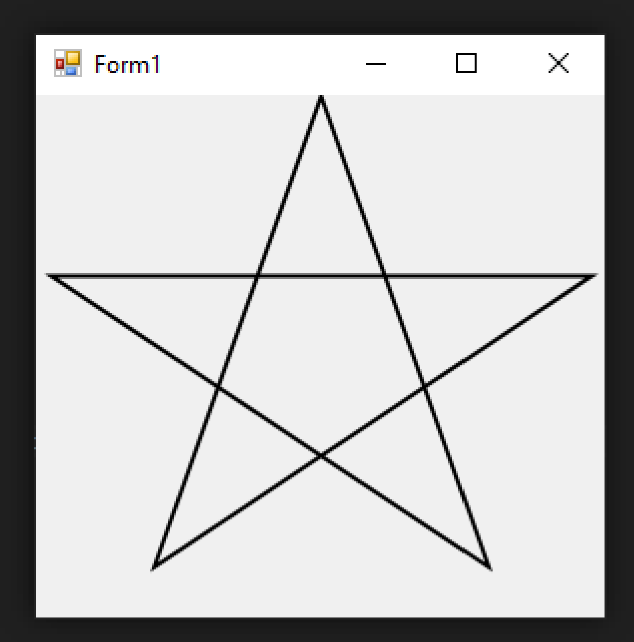
```
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;
            int nPoints = 5; Rectangle bounds = this.ClientRectangle;

            PointF[] points = new PointF[nPoints];

            double rx = bounds.Width / 2;
            double ry = bounds.Height / 2;
            double cx = bounds.X + rx;
            double cy = bounds.Y + ry;

            // Start at the top.
            double theta = -Math.PI / 2;
            double radtheta = 4 * Math.PI / nPoints;
            for (int i = 0; i < nPoints; i++)
            {
                points[i] = new PointF(
                    (float)(cx + rx * Math.Cos(theta)),
                    (float)(cy + ry * Math.Sin(theta)));
                theta += radtheta;
            }

            e.Graphics.DrawPolygon(new Pen(Color.Black, 2), points);
        }
    }
}
```



رسم النصوص

لست مضطر للتعامل مع الأدوات الجاهزة – مثل Label – لطباعة النصوص على منطقة الرسم، فكائنات الرسومات توفر لك رسم النصوص، بإمكانيات كبيرة. يمكنك من خلال رسم النصوص كتابة محتوى نصي داخل منطقة رسم معينة، عوضا عن استخدام أدوات جاهزة، وبالتالي استقلالية الأدوات أكثر.



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

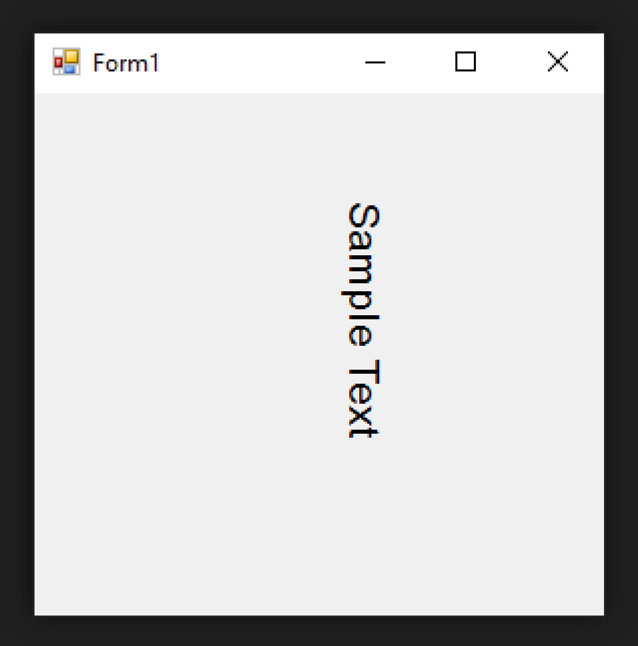
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            // إنشاء نص لرسمه
            string s = "Sample Text";

            // إنشاء الخط واللون
            Font f = new Font("Arial", 16);
            SolidBrush b = new SolidBrush(Color.Black);

            // إحداثيات الرسم (الزاوية العليا اليسرى)
            float x = 150.0F;
            float y = 50.0F;

            // إنشاء كائن ينسق العبارات النصية قبل رسمها
            StringFormat sf = new StringFormat();
            sf.FormatFlags = StringFormatFlags.DirectionVertical;

            // رسم القيمة النصية بالاعتبارات التي اخترناها
            e.Graphics.DrawString(s, f, b, x, y, sf);
        }
    }
}
```



فضلًا عن رسم النصوص بشكل ثنائي الأبعاد بوضعيات مختلفة، يمكنك رسم نص بتأثير ثلاثي الأبعاد أو رسم نص شفاف على صورة (علامة مائية Watermark)، فإذا أردت استكشاف المزيد راجع كتاب م. محمد حمدي غانم "الرسم والتلوين باستخدام +GDI وDirectX لمبرمجي سي شارب".

رسم نص داخل مستطيل

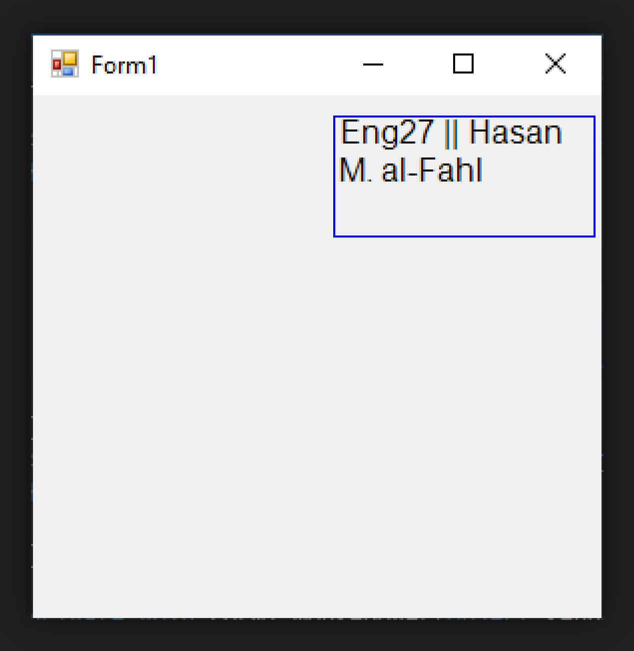


```
uusing System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            Rectangle r = new Rectangle(150, 10, 130, 60);
            g.DrawRectangle(Pens.Blue, r);
            g.DrawString("Eng27 || Hasan M. al-Fahl",
                new Font("Arial", 12), Brushes.Black, r);
        }
    }
}
```



رسم نص على محيط دائرة



```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            BackColor = SystemColors.Window;
            ForeColor = SystemColors.WindowText;
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            SetScale(g);
            DrawFace(g);
            base.OnPaint(e);
        }

        private void SetScale(Graphics g)
        {
            g.TranslateTransform(Width / 2, Height / 2);

            float inches = Math.Min(Width / g.DpiX, Height / g.DpiX);

            g.ScaleTransform(inches * g.DpiX / 2000, inches * g.DpiY / 2000);
        }
```

```
        private void DrawFace(Graphics g)
        {
            Brush brush = new SolidBrush(ForeColor);
            Font font = new Font("Arial", 40);

            float x, y;

            const int numHours = 12;
            const int deg = 360 / numHours;
            const int FaceRadius = 450;

            for (int i = 1; i <= numHours; i++)
            {
                x = GetCos(i * deg + 90) * FaceRadius;
                y = GetSin(i * deg + 90) * FaceRadius;

                StringFormat format = new StringFormat();
                format.Alignment = StringAlignment.Center;
                format.LineAlignment = StringAlignment.Center;

                g.DrawString(i.ToString(), font, brush, -x, -y, format);
            }

            brush.Dispose();
            font.Dispose();
        }

        private static float GetSin(float degAngle)
        { return (float)Math.Sin(Math.PI * degAngle / 180f); }

        private static float GetCos(float degAngle)
        { return (float)Math.Cos(Math.PI * degAngle / 180f); }
    }
}
```



على اعتبار عملية الرسم تمت في الحدث Paint، فإن الرسم سيتم تحديثه في كل مرة يحدث فيها تحديث لمنطقة الرسم، لاحظ ماذا سيحدث إذا ما تم تكبير النافذة:

رسم نص محاذى نحو اليسار



```
using System.Drawing;
using System.Windows.Forms;

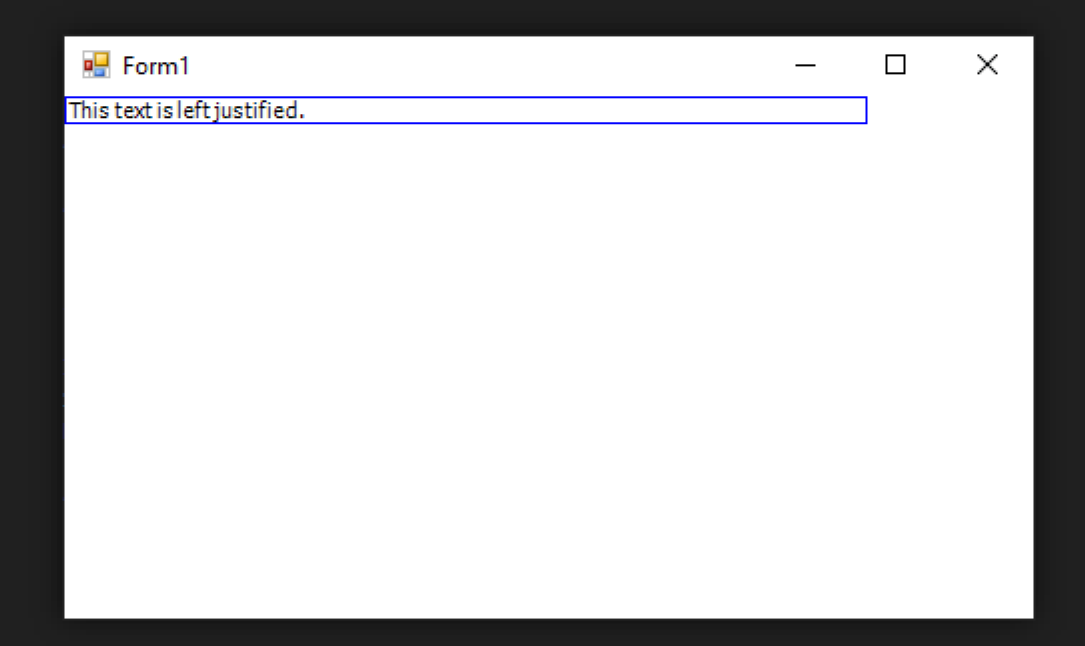
namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            SetStyle(ControlStyles.Opaque, true);
            Bounds = new Rectangle(0, 0, 500, 300);
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            int y = 0;

            g.FillRectangle(Brushes.White, ClientRectangle);

            // draw left justified text
            Rectangle rect = new Rectangle(0, y, 400, Font.Height);
            g.DrawRectangle(Pens.Blue, rect);
```

```
            g.DrawString("This text is left justified.",
                Font, Brushes.Black, rect);
            y += Font.Height + 20;
        }
    }
}
```



رسم نص محاذى نحو اليمين



```
using System.Drawing;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            SetStyle(ControlStyles.Opaque, true);
            Bounds = new Rectangle(0, 0, 500, 300);
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            int y = 0;

            g.FillRectangle(Brushes.White, ClientRectangle);

            Rectangle rect = new Rectangle(0, y, 400, Font.Height);
            g.DrawRectangle(Pens.Blue, rect);
```

```
            StringFormat sf = new StringFormat();
            sf.Alignment = StringAlignment.Far;
            g.DrawString("This text is right justified.",
                Font, Brushes.Blue, rect, sf);
            y += Font.Height + 20;
        }
    }
}
```

Form1

This text is right justified.

رسم نص محاذى نحو الوسط



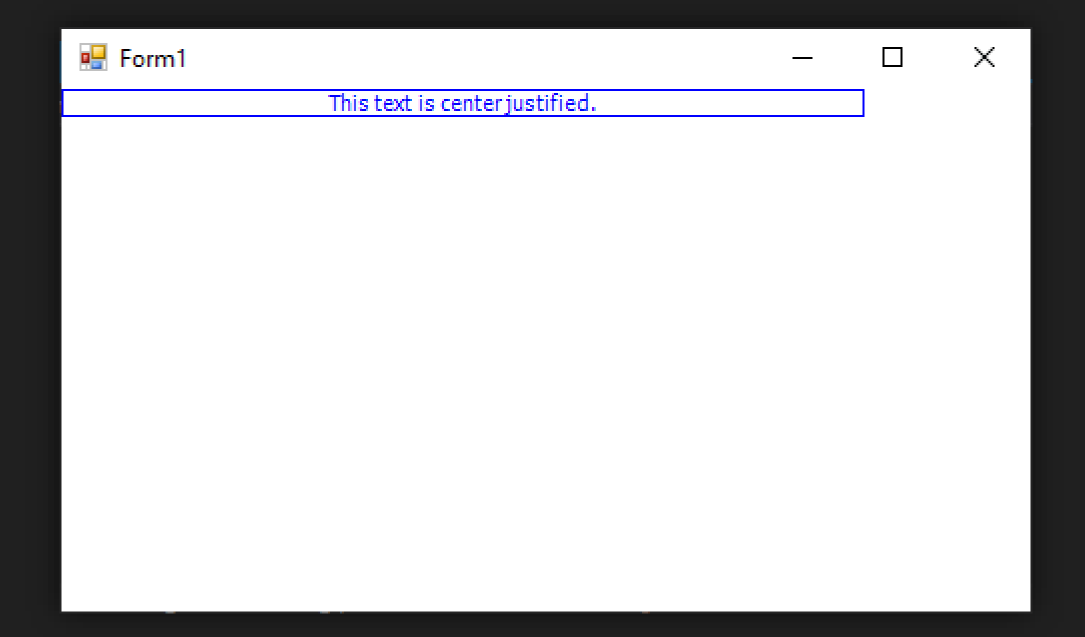
```
using System.Drawing;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            SetStyle(ControlStyles.Opaque, true);
            Bounds = new Rectangle(0, 0, 500, 300);
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            int y = 0;

            g.FillRectangle(Brushes.White, ClientRectangle);
```

```
                Rectangle rect = new Rectangle(0, y, 400, Font.Height);
                g.DrawRectangle(Pens.Blue, rect);
                StringFormat sf = new StringFormat();
                sf.Alignment = StringAlignment.Center;
                g.DrawString("This text is center justified.",
                    Font, Brushes.Blue, rect, sf);
                y += Font.Height + 20;
            }
        }
}
```



قياس النصوص

يمكنك قياس حجم النصوص (ارتفاعها وعرضها) من خلال الطريقة MeasureString، وبالتالي يمكنك محاذاة الرسوم الأخرى عند رسمها مع النصوص، إذ إنه من الصعب التنبؤ بارتفاع وعرض الرسومات النصية، والتي تتغير مع كل خط.

المناطق والمسارات

تعطيك المسارات إمكانية رسم مجموعة من الأشكال في الوقت نفسه، وذلك بإضافة هذه الأشكال من خلال طرق مناظرة لتلك الطرق الموجودة في الفئة Graphics. فمثلًا يمكنك رسم خط من خلال الطريقة DrawLine التابعة للفئة Graphics، ويمكنك إضافة

الخط إلى مسار ما من خلال الطريقة AddLine. بالمثل يمكنك رسم مستطيل من خلال الطريقة DrawRectangle ويمكنك إضافته لمسار ما من خلال الطريقة AddRectangle، وهكذا. يمكنك إنشاء كائن مسار باستنساخ الفئة GraphicsPath.

تختلف طرق المسارات عن طرق الرسومات Graphics بالوسيط الأول فقط، فطرق الرسومات تأخذ في البداية كائنًا من النوع Pen أو كائنًا من النوع Brush، في حين أن طرق المسارات لا تأخذ أيًّا منهما، فمن خلال المسارات أنت فقط تشكل الأشكال التي تود رسمها، ولكن لا ترسمها. أما بقية الوسطاء فهي ذاتها في طرق الرسومات وطرق المسارات.

لرسم مسار ما استدع الطريقة DrawPath التابعة للفئة Graphics.

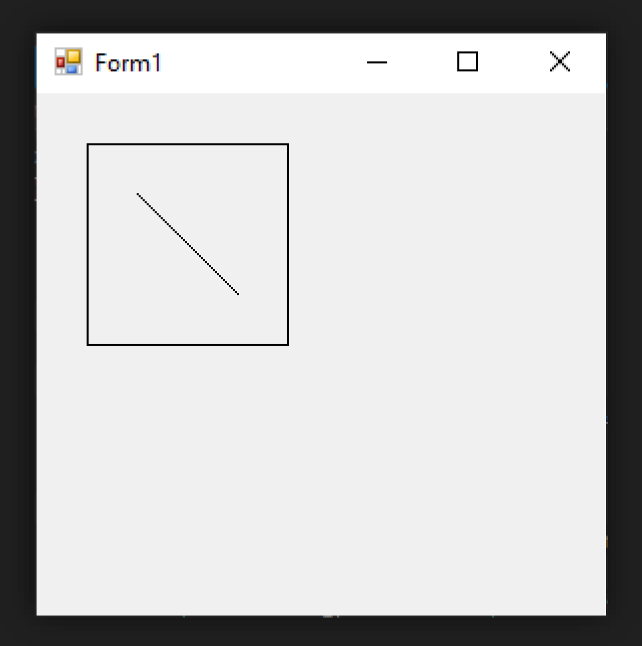
لاحظ المثال:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            GraphicsPath gp = new GraphicsPath();
            gp.AddLine(50, 50, 100, 100);
            gp.AddRectangle(new Rectangle(25, 25, 100, 100));
            e.Graphics.DrawPath(Pens.Black, gp);
        }
    }
}
```



اقتطاع الرسوم Clipping



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

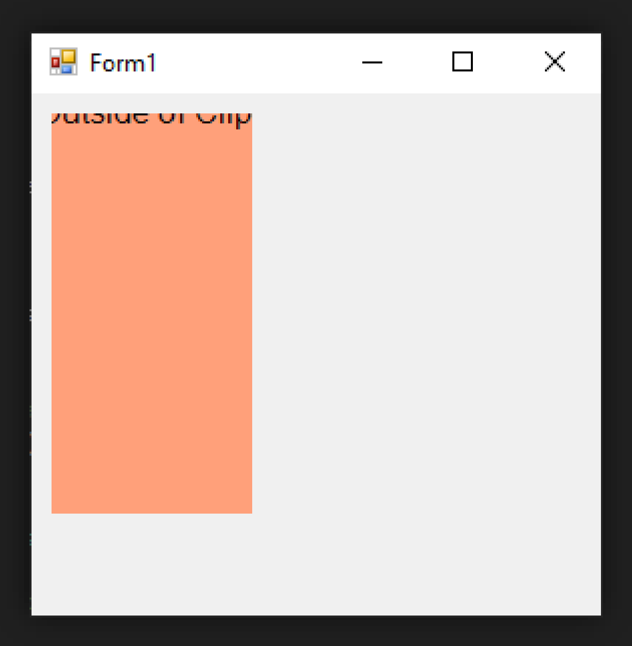
namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {

            // ضبط منطقة الاقتطاع على منطقة جديدة
            e.Graphics.Clip = new Region(new Rectangle(10, 10, 100, 200));

            // إملاء المنطقة بلون
            e.Graphics.FillRegion(Brushes.LightSalmon, e.Graphics.Clip);

            // رسم قيمة نصية
            e.Graphics.DrawString("Outside of Clip", new Font("Arial",
                12.0F, FontStyle.Regular), Brushes.Black, 0.0F, 0.0F);
        }
    }
}
```



يمكنك اقتطاع الرسوم بمسار ما:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Point[] polyPoints = {
                                    new Point(10, 10),
                                    new Point(150, 10),
                                    new Point(100, 75),
                                    new Point(100, 150)};

            GraphicsPath path = new GraphicsPath();
            path.AddPolygon(polyPoints);

            // إنشاء منطقة على أساس المسار
            Region region = new Region(path);

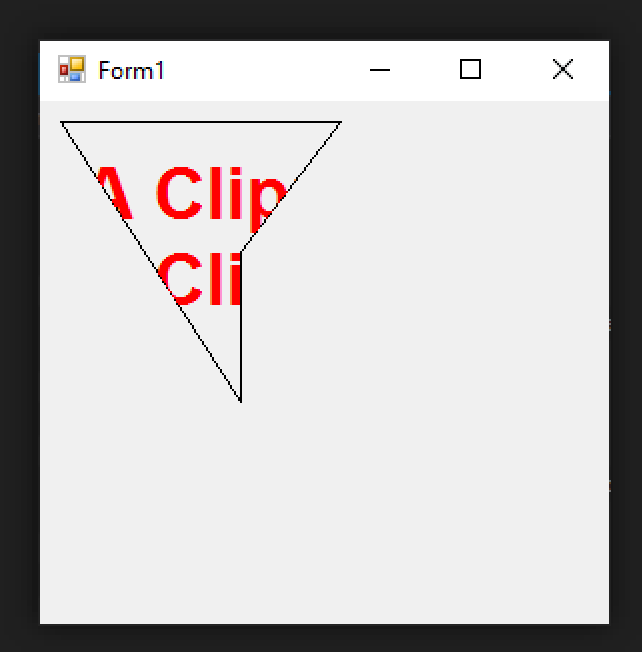
            // رسم حواف المسار
            Pen pen = Pens.Black;
            e.Graphics.DrawPath(pen, path);

            // اقتطاع الرسم
            e.Graphics.SetClip(region, CombineMode.Replace);
```

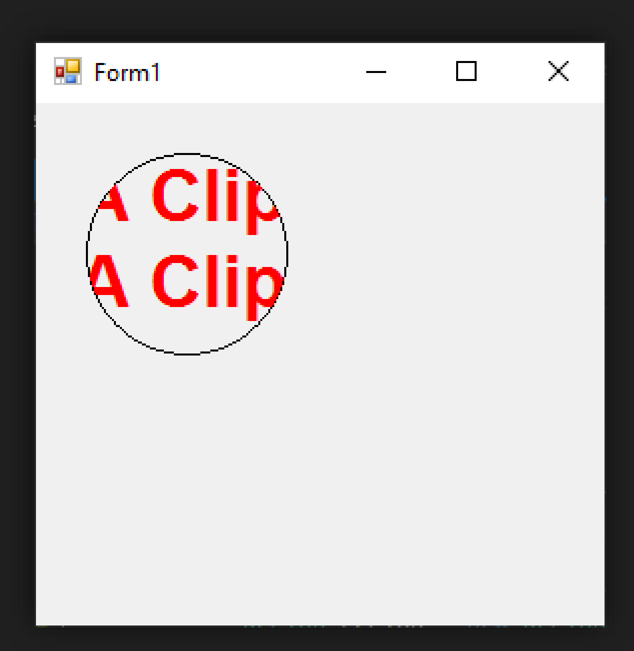
```
            // رسم نص مقطوع
            FontFamily fontFamily = new FontFamily("Arial");
            Font font = new Font(
               fontFamily,
               36, FontStyle.Bold,
               GraphicsUnit.Pixel);
            SolidBrush solidBrush = new SolidBrush(
                Color.FromArgb(255, 255, 0, 0));

            e.Graphics.DrawString(
               "A Clipping Region",
               font, solidBrush,
               new PointF(15, 25));

            e.Graphics.DrawString(
               "A Clipping Region",
               font,
               solidBrush,
               new PointF(15, 68));
        }
    }
}
```



وبإضافة دائرة قطرها 100 وزاوية المربع المحيط بها اليسرى تقع في النقطة (25, 25) عوضًا عن المضلع:



تقاطع مناطق الرسم



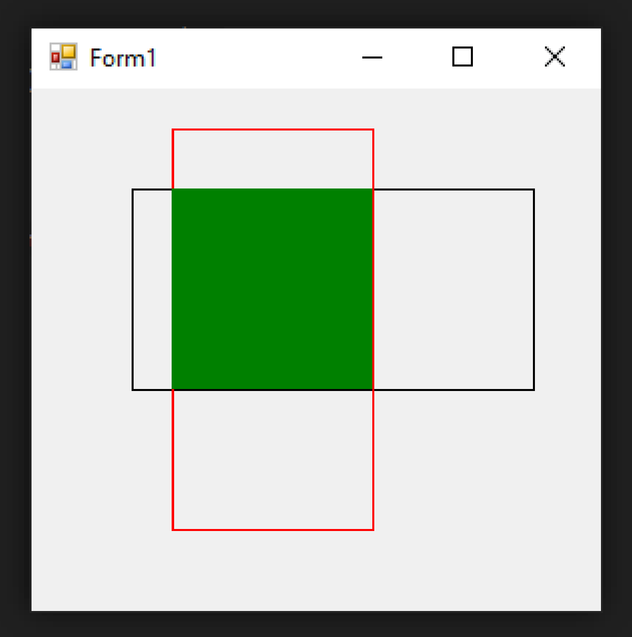
```
using System.Drawing;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Rectangle rec1 = new Rectangle(50, 50, 200, 100);
            Rectangle rec2 = new Rectangle(70, 20, 100, 200);

            e.Graphics.DrawRectangle(Pens.Black, rec1);
            e.Graphics.DrawRectangle(Pens.Red, rec2);

            if (rec1.IntersectsWith(rec2))
            {
                rec1.Intersect(rec2);
                if (!rec1.IsEmpty)
                {
                    e.Graphics.FillRectangle(Brushes.Green, rec1);
                } } } } }
```



مثال آخر:



```
using System.Drawing;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

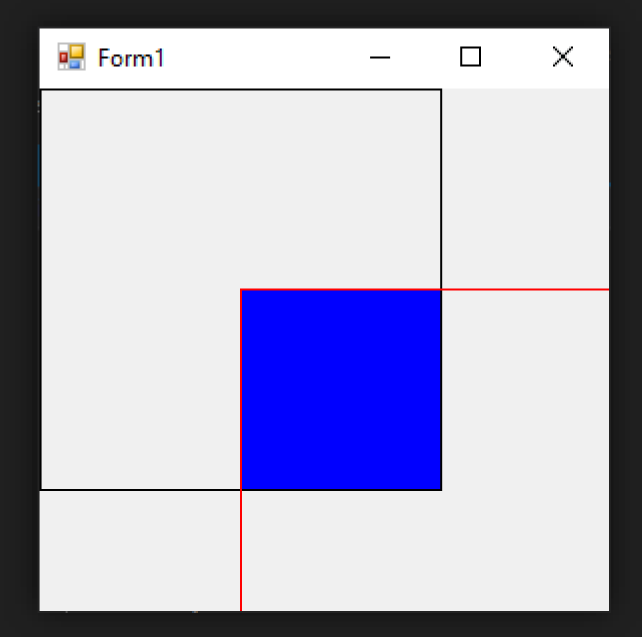
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Rectangle clipRect = new Rectangle(0, 0, 200, 200);
            e.Graphics.SetClip(clipRect);

            Rectangle intersectRect = new Rectangle(100, 100, 200, 200);
            e.Graphics.IntersectClip(intersectRect);

            e.Graphics.FillRectangle(new SolidBrush(Color.Blue), 0, 0, 500, 500);

            e.Graphics.ResetClip();

            e.Graphics.DrawRectangle(new Pen(Color.Black), clipRect);
            e.Graphics.DrawRectangle(new Pen(Color.Red), intersectRect);
        }
    }
}
```



إغلاق الرسومات تلقائيًّا



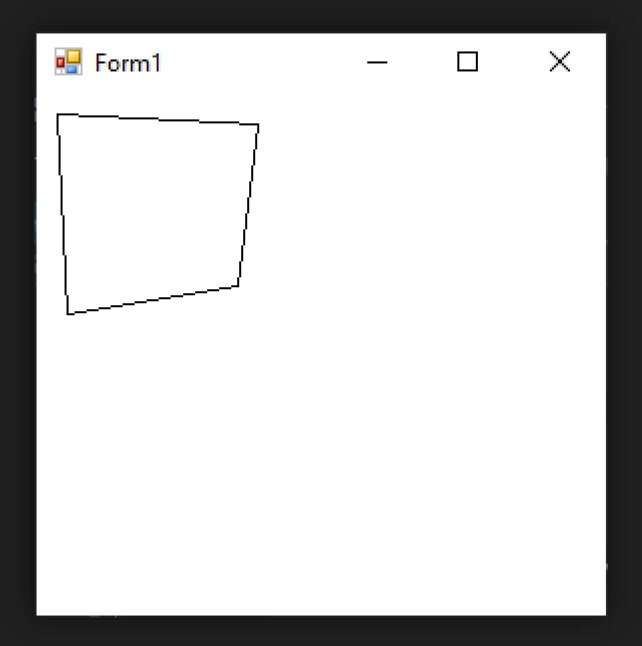
```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            GraphicsPath gp = new GraphicsPath();

            gp.AddLine(10, 10, 110, 15);
            gp.AddLine(110, 15, 100, 96);
            gp.AddLine(100, 96, 15, 110);
            gp.CloseFigure();

            g.FillRectangle(Brushes.White, this.ClientRectangle);
            g.DrawPath(Pens.Black, gp);
            gp.Dispose();
        }
    }
}
```



لولا الإجراء CloseFigure لما تم إغلاق الشكل.

تظليل رسم عند النقر داخل مسار ما



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        GraphicsPath myPath = new GraphicsPath();
        private bool isImageClicked = false;
        private int imageClickedIndex;

        public Form1()
        {
            InitializeComponent();
            myPath.StartFigure();
            Point point1 = new Point(100, 100);
            Point point2 = new Point(100, 200);
            Point point3 = new Point(200, 100);
            Point point4 = new Point(200, 200);
            Point[] points = { point1, point2, point3, point4 };
            myPath.AddCurve(points);
            myPath.CloseFigure();
        }

        private void Form1_MouseUp(object sender, MouseEventArgs e)
        {
            Point mousePt = new Point(e.X, e.Y);
```

```
            if (myPath.IsVisible(mousePt))
            {
                isImageClicked = true;
                imageClickedIndex = 3;
            }
            else isImageClicked = false;

            Invalidate();
        }

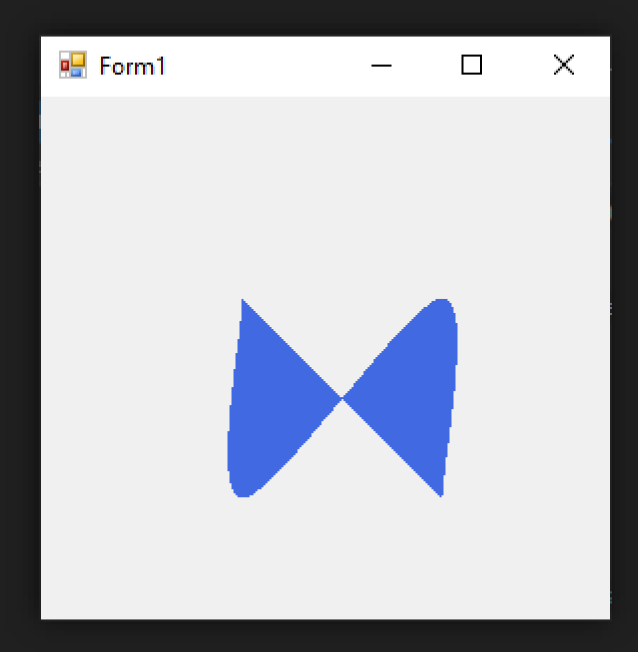
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.FillPath(Brushes.RoyalBlue, myPath);

            if (isImageClicked == true)
            {
                Pen outline = new Pen(Color.Purple, 5);

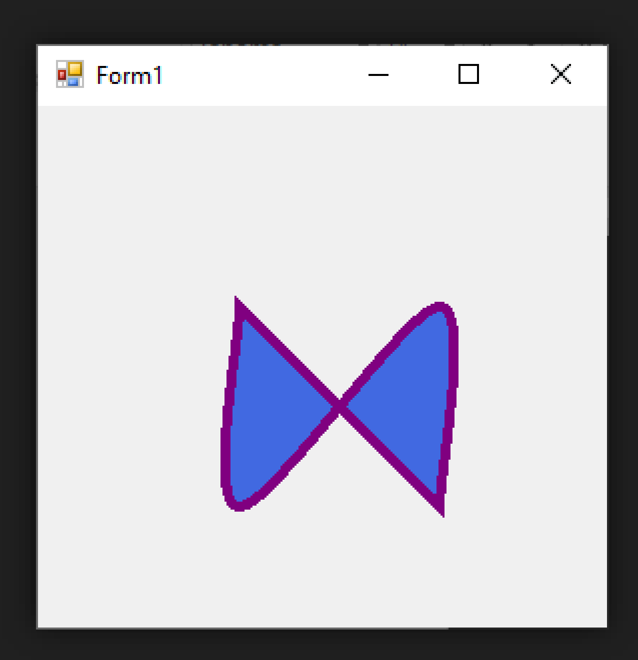
                switch (imageClickedIndex)
                {
                    case 3:
                        g.DrawPath(outline, myPath);
                        break;

                    default: break;
                }
            }
        }
    }
}
```

عند تشغيل البرنامج:



عند النقر على المنطقة المظللة:



التحقق من أن مؤشر الفأرة خارج أو داخل منطقة ما



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

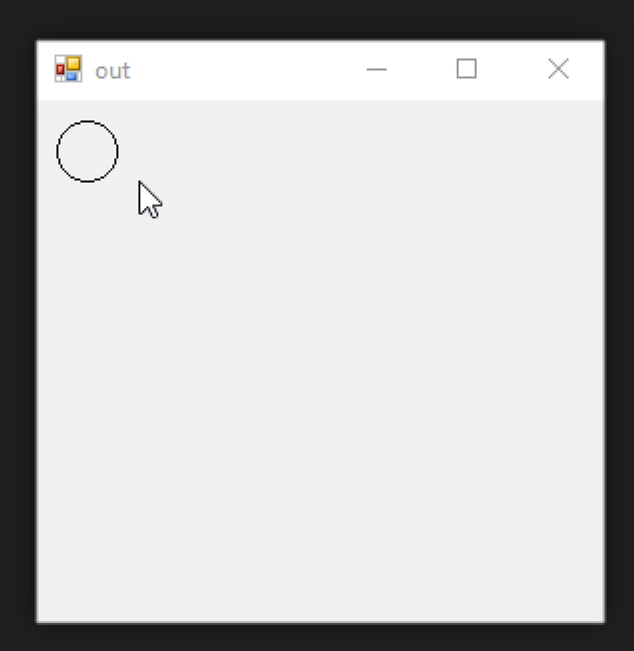
namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        GraphicsPath gP;

        public Form1()
        {
            InitializeComponent();
            gP = new GraphicsPath();
            gP.AddEllipse(10, 10, 30, 30);
        }

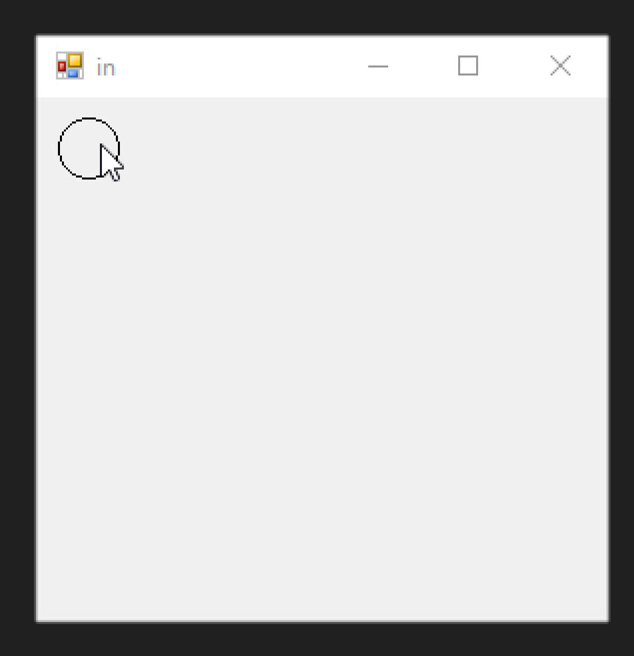
        protected override void OnPaint(System.Windows.Forms.PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.DrawPath(Pens.Black, gP);
            g.Dispose();
        }

        private void Form1_MouseMove(object sender, MouseEventArgs e)
        {
            Region reg = new Region(gP);
            if (reg.IsVisible(new Point(e.X, e.Y)))
                Text = "in";
```

```
                else
                    Text = "out";
            }
        }
}
```



بعد المرور داخل منطقة الرسومات:

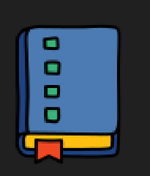


التلوين، بشكل متقدم

يمكنك تلوين أشكالك بشكل متدرج، خطيًّا أو متشعبًا، وذلك كما سنرى في الفقرات التالية.

التلوين المتدرج الخطي

التلوين المتدرج الخطي يقتضي وجود لون بداية ولون نهاية وما بينهما مزيج منهما، والانتقال بينهما يكون خطيًا متغيرًا بانتظام. الفئة LinearGradiantBrush تعطيك فرش تلوّن أشكالك بلون متدرج وفق الصيغة:



```
LinearGradientBrush lgb = new LinearGradientBrush(نقطة1,
    نقطة2,
    لون_البداية,
    لون_النهاية);
```

كما يمكنك إنشاء كائن من هذه الفئة بالصيغة:



```
LinearGradientBrush lgb = new LinearGradientBrush(مستطيل,
    لون_البداية,
    لون_النهاية,
    زاوية);
```

أو باستبدال الوسيط الممثل للزاوية في الصيغة الأخيرة بمعدّد من النوع LinearGradiantMode، والذي يحدد اتجاه التدرج (مائل بزاوية 45 من اليمين إلى اليسار، أو بالعكس، أو أفقي، أو شاقولي). كما يمكنك تحديد زاوية التدرج بالصيغة الأخيرة، مما يتيح لك زوايا أكثر من تلك التي ستحصل عليها عند إرسال المعدد المذكور. وبالمثل فإن الصيغة الأولى تحدد اتجاه التدرج على أساس مواقع النقاط بالنسبة لبعضها.

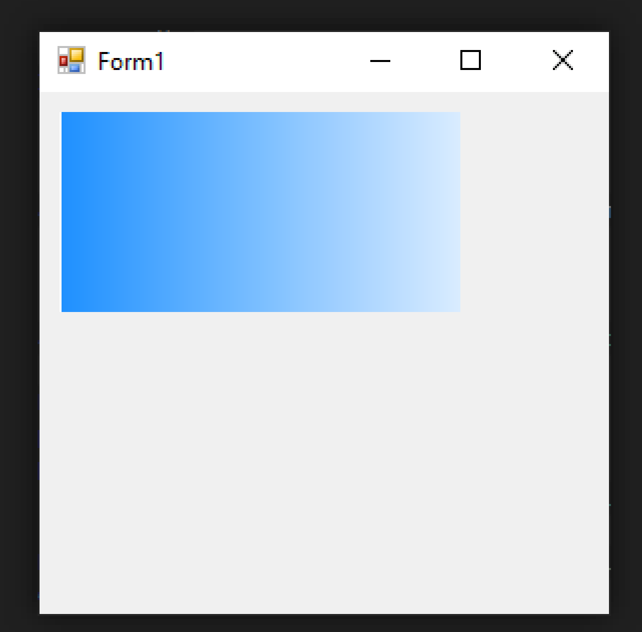
يتم تحديد موقع اللون الأول واللون الثاني اعتمادًا على مواقع النقاط عند استخدام الصيغة الأولى، وعلى زوايا المستطيل عند استخدام الصيغة الثانية.



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

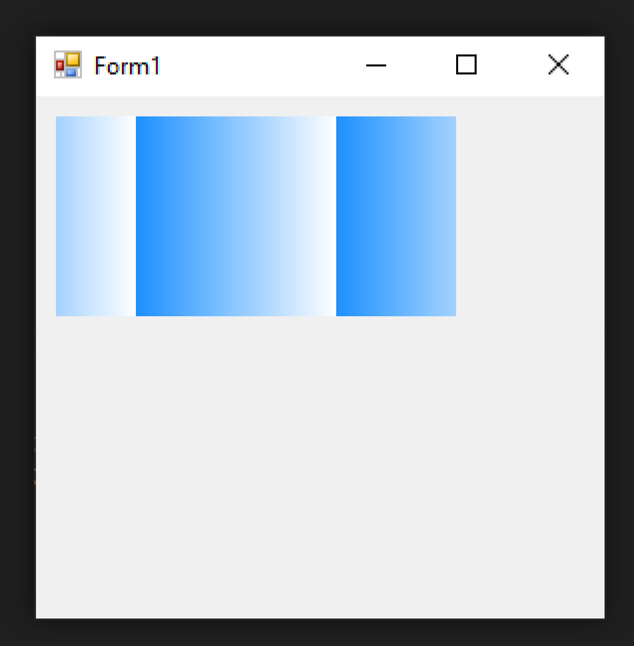
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Point p1, p2;
            p1 = new Point(10, 0);
            p2 = new Point(250, 0);
            LinearGradientBrush lgd = new LinearGradientBrush(p1,
                p2, Color.DodgerBlue, Color.White);
            Rectangle rec = new Rectangle(10,10,200,100);
            e.Graphics.FillRectangle(lgd, rec);
        }
    }
}
```



لاحظ أن النقاط متعلقة بإحداثيات منطقة الرسم وليس الشكل الذي يتم رسمه/إملاؤه.

لاحظ أيضًا أننا تعمّدنا وجود نقطة بداية التدرج قبل أو في بداية الشكل الذي نرغب بتدريج لونه، ونقطة نهاية التدرج بعد أو في نهاية الشكل الذي نرغب بتدريج لونه، وإلا سيتم

تكرار التدريج عدة مرات حتى ينتهي الشكل، لاحظ ما يحدث إذا وضعنا النقطة الأولى بعد بداية الشكل والنقطة الثانية قبل نهاية الشكل:



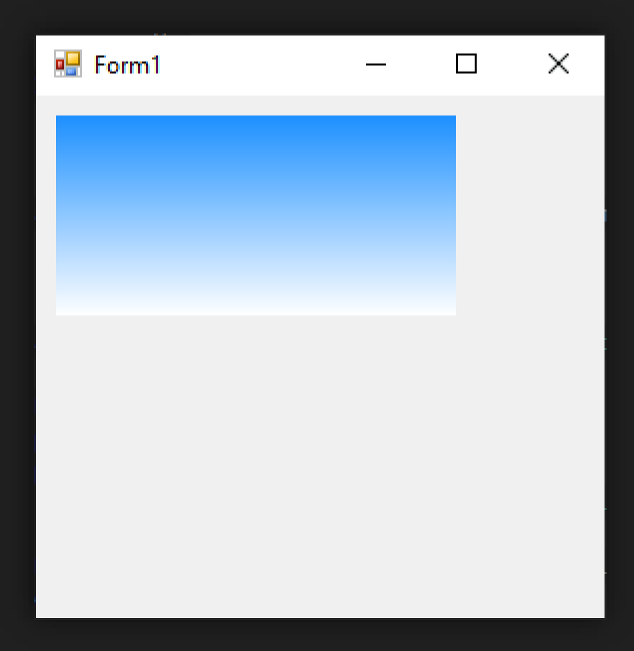
يمكنك أن تجعل التدريج من الأعلى للأسفل بتثبيت الفواصل وتغيير التراتيب:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Point p1, p2;
            p1 = new Point(0, 10);
            p2 = new Point(0, 110);
            LinearGradientBrush lgd = new LinearGradientBrush(p1,
                p2, Color.DodgerBlue, Color.White);
            Rectangle rec = new Rectangle(10,10,200,100);
            e.Graphics.FillRectangle(lgd, rec);
        }
    }
}
```



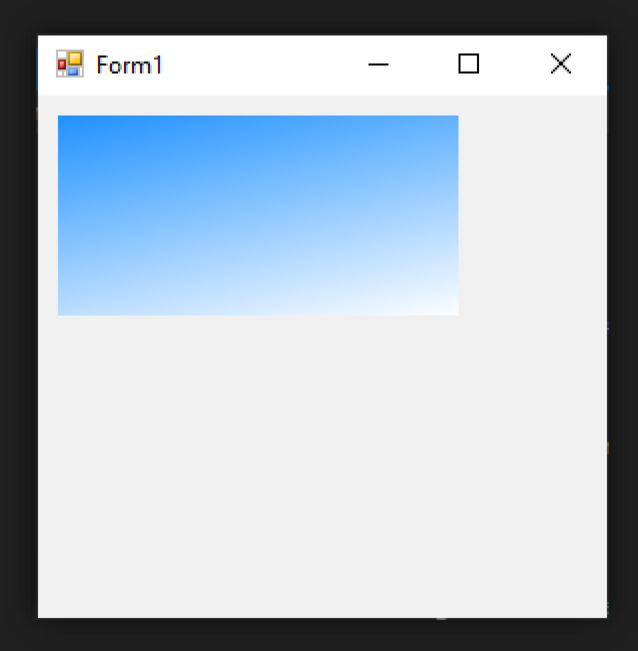
ولتحصل على تدريج مائل، قم بتغيير الفواصل والتراتيب معًا:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Point p1, p2;
            p1 = new Point(0, 10);
            p2 = new Point(30, 150);
            LinearGradientBrush lgd = new LinearGradientBrush(p1,
                p2, Color.DodgerBlue, Color.White);
            Rectangle rec = new Rectangle(10,10,200,100);
            e.Graphics.FillRectangle(lgd, rec);
        }
    }
}
```



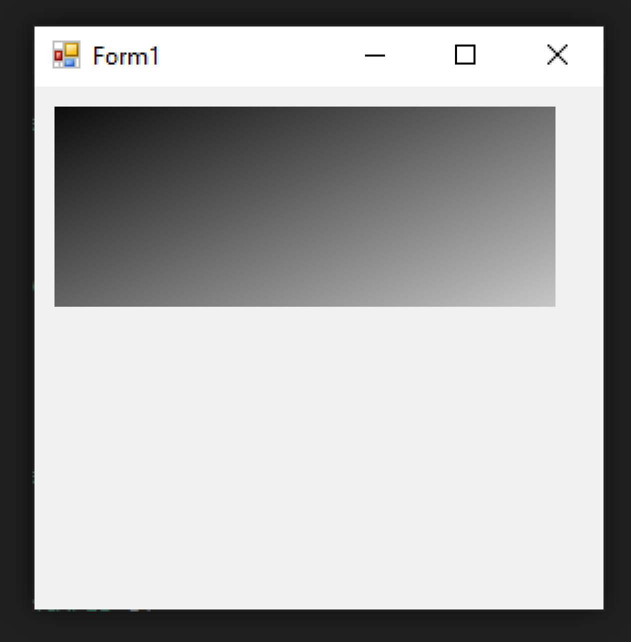
أما عند استخدام الصيغة الثانية:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            int width, height;
            width = 250;
            height = 100;
            Brush brGradient = new LinearGradientBrush(
                new Rectangle(0, 0, width +20, height +20),
                Color.Black, Color.LightGray, 45, true);
            g.FillRectangle(brGradient, 10, 10,width , height);
        }
    }
}
```



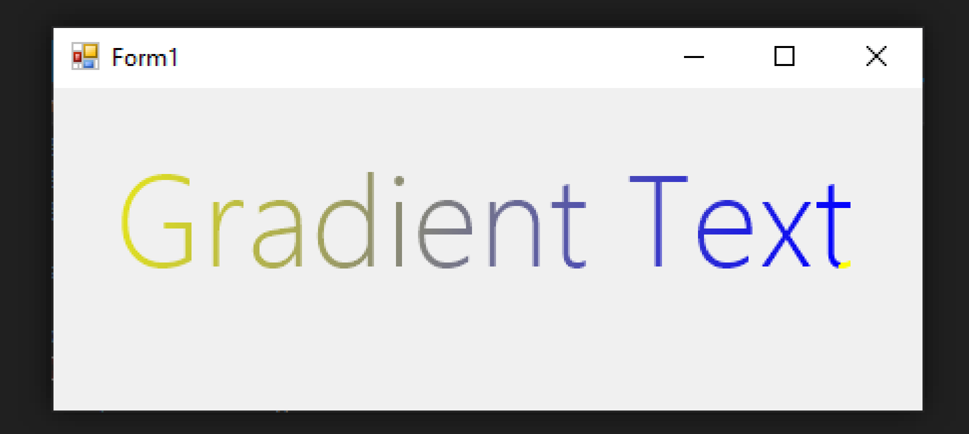
بشكل مشابه يمكنك تلوين النصوص بلون متدرج:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            Width = 450;
            Height = 200;
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.TextRenderingHint = System.Drawing.Text.TextRenderingHint.AntiAlias;
            Font f = new Font("Segoe UI Light", 48, FontStyle.Regular,
                GraphicsUnit.Point);
            PointF gradientStart = new PointF(0, 0);
            string txt = "Gradient Text";
            SizeF txtSize = g.MeasureString(txt, f);
            PointF gradientEnd = new PointF(txtSize.Width, txtSize.Height);
            LinearGradientBrush grBrush = new LinearGradientBrush(gradientStart,
                gradientEnd, Color.Yellow, Color.Blue);
            g.DrawString(txt, f, grBrush, 20, 20);
        }
    }
}
```



بإمكانك تصحيح ما يسمى بالـ غامّا Gamma (شيء يجعل الصور أقرب للواقع) من خلال تفعيل خاصية GammaCorrection:

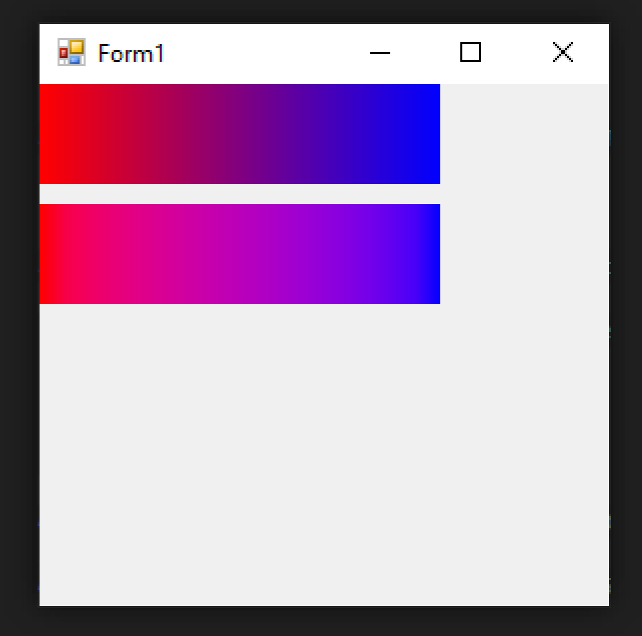


```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            LinearGradientBrush linGrBrush = new LinearGradientBrush(
                new Point(0, 10),
                new Point(200, 10),
                Color.Red,
                Color.Blue);

            e.Graphics.FillRectangle(linGrBrush, 0, 0, 200, 50);
            linGrBrush.GammaCorrection = true;
            e.Graphics.FillRectangle(linGrBrush, 0, 60, 200, 50);
        }
    }
}
```



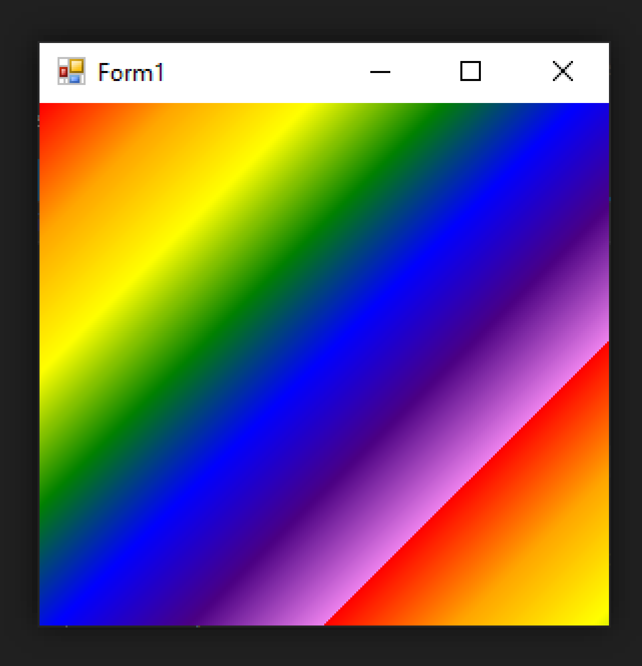
إضافة ألوان أكثر:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.ResizeRedraw = true;
        }

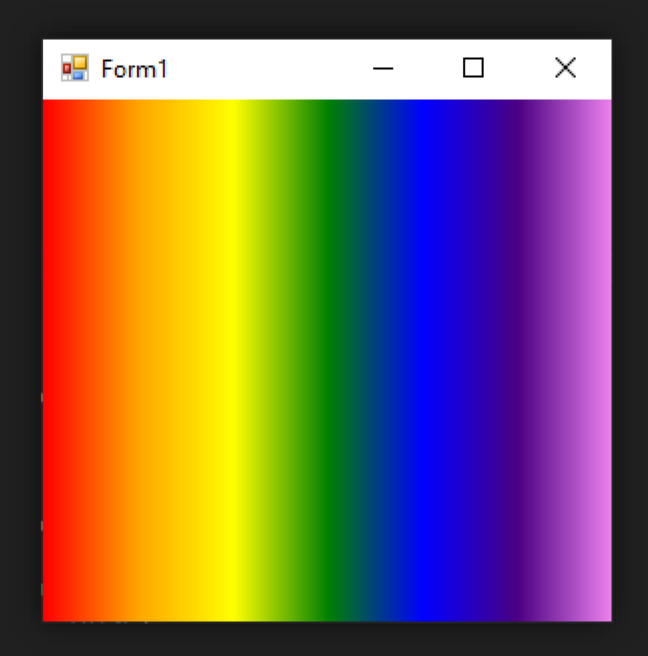
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            LinearGradientBrush br = new LinearGradientBrush(this.ClientRectangle,
                Color.Black, Color.Black, 0, false);
            ColorBlend cb = new ColorBlend();
            cb.Positions = new[] { 0, 1 / 6f, 2 / 6f, 3 / 6f, 4 / 6f, 5 / 6f, 1 };
            cb.Colors = new[] { Color.Red, Color.Orange, Color.Yellow,
                Color.Green, Color.Blue, Color.Indigo, Color.Violet };
            br.InterpolationColors = cb;
            br.RotateTransform(45);
            e.Graphics.FillRectangle(br, this.ClientRectangle);
        }
    }
}
```



بتكبير النافذة (بمعنى آخر: بتحفيز الحدث Paint):



وبحذف السطر الذي يدوّر المحاور RotateTransform:





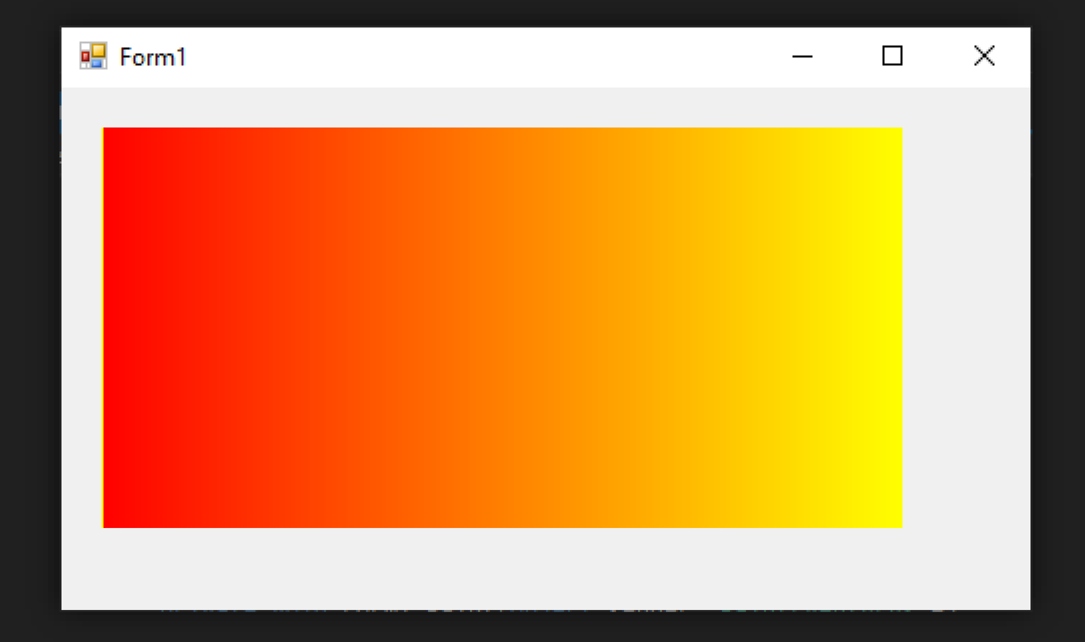
يمكنك الاعتماد على المسارات لإنشاء تدرج في اللون:



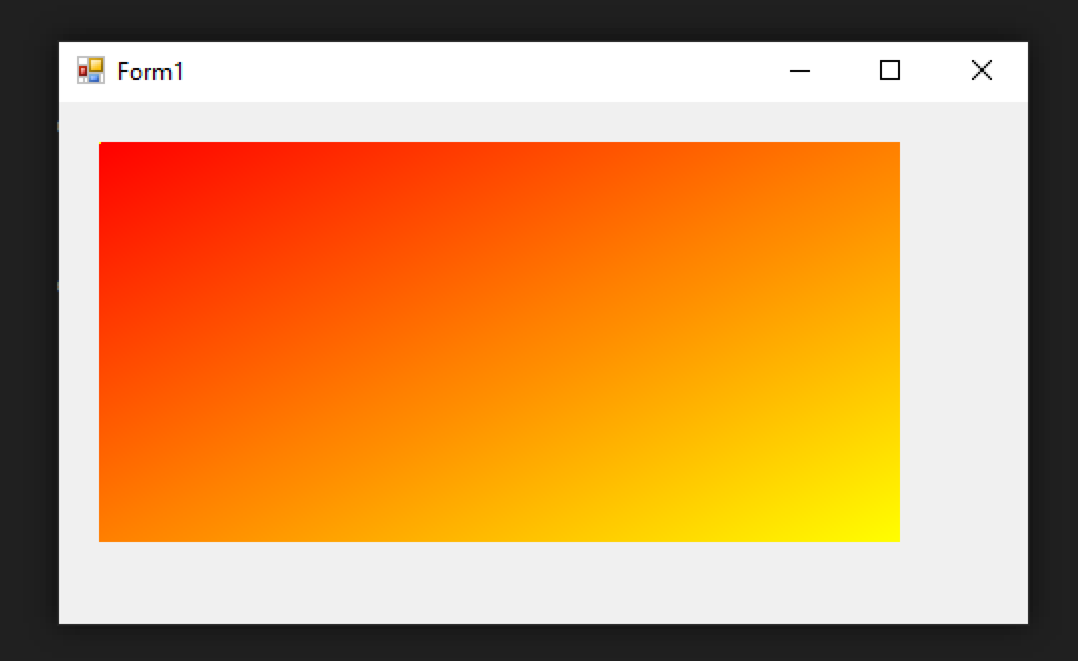
```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            Width = 500;
        }
```

```
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            GraphicsPath path = new GraphicsPath();
            Rectangle rec = new Rectangle(20, 20, 400, 200);
            path.AddRectangle(rec);
            LinearGradientBrush lBrush = new LinearGradientBrush(rec,
                Color.Red, Color.Yellow,
                LinearGradientMode.Horizontal);
            g.FillPath(lBrush, path);
        }
    }
}
```



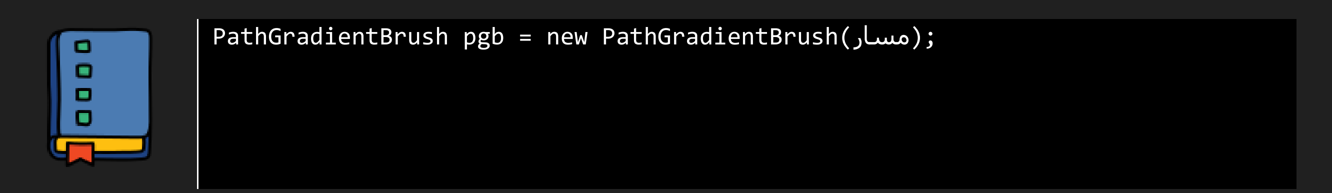
وبجعل وضع التدرج الخطي LinearGradientMode مائلًا:



الفقرة التالية فيها المزيد عن المسارات.

التلوين المتدرج المتشعب

تعتمد الفرش متدرجة اللون المتشعب على المسارات، لذلك فيمكنك اعتبار هذه الفقرة إحدى تطبيقات المسارات. مبدأ هذه الفرش هو الانتقال من لون ما إلى لون آخر (أو ألوان أخرى)، بحيث ينتشر اللون باتجاهات مختلفة وليس باتجاه واحد كما هو الحال مع الفرش المتدرجة الخطية. يمكنك إنشاء كائن من فرشاة متدرجة متشعبة باستنساخ الفئة PathGradientBrush كما في الصيغة التالية:



```
PathGradientBrush pgb = new PathGradientBrush(مسار);
```

تعتمد الفرش المتدرجة المتشعبة هذه على لون مركزي CenterColor ومجموعة من الألوان المحيطة SurroundColors والتي يتم إدخالها على شكل مصفوفة من الألوان.

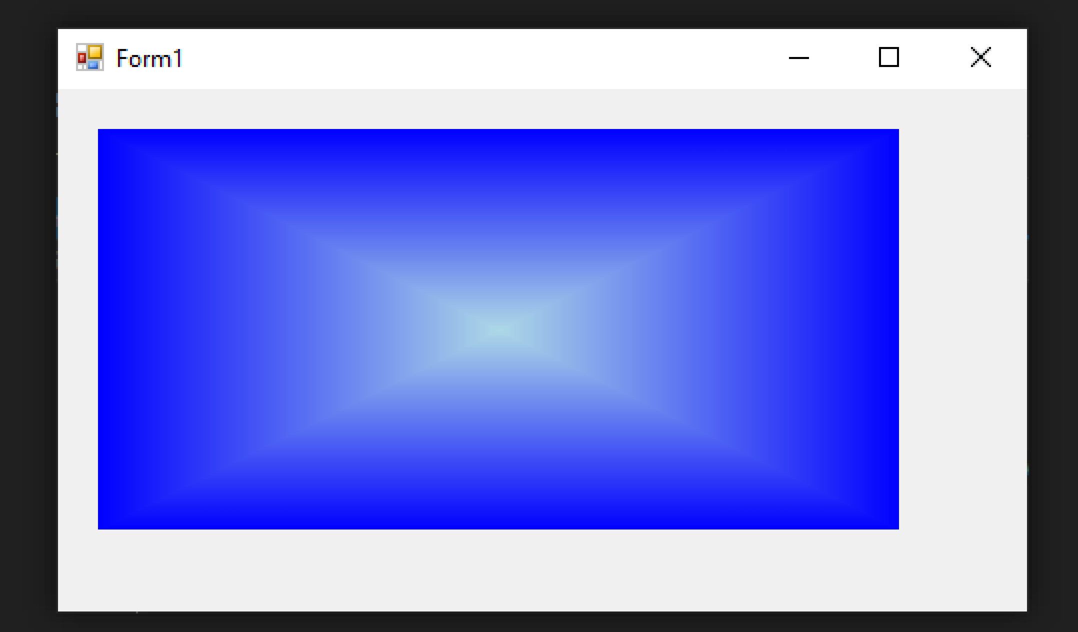
لاحظ المثال:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            Width = 500;
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            GraphicsPath path = new GraphicsPath();
            Rectangle rec = new Rectangle(20, 20, 400, 200);
            path.AddRectangle(rec);
            PathGradientBrush pathBrush = new PathGradientBrush(path);
            Color centerColor = Color.LightBlue;
            Color[] surroundColors = { Color.Blue };
            pathBrush.CenterColor = centerColor;
            pathBrush.SurroundColors = surroundColors;
            g.FillPath(pathBrush, path);
        }
    }
}
```



وبوضع ألوان أكثر وإضافة بعض الخطوط للمسار:



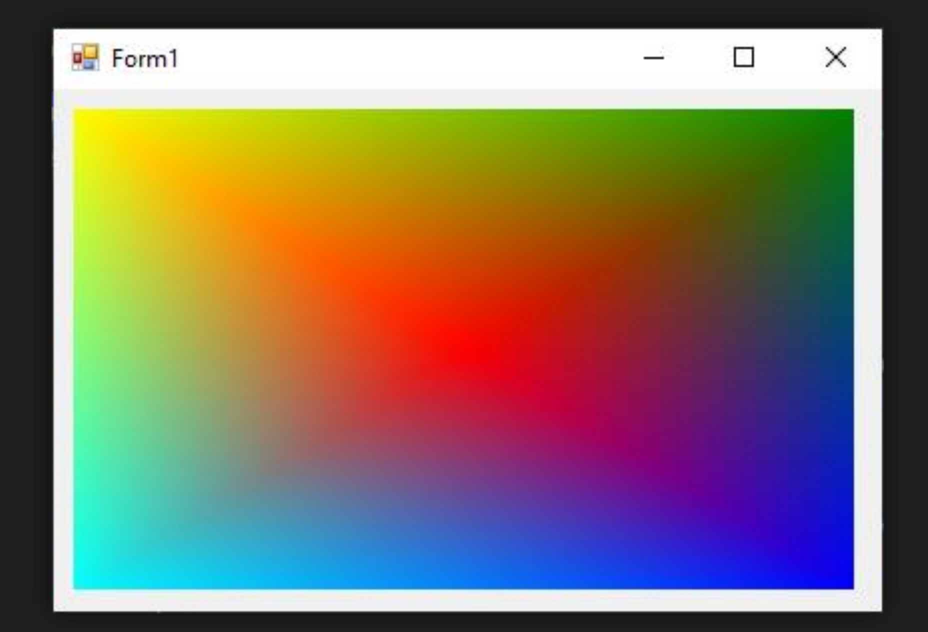
```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); Width = 430; }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            GraphicsPath path = new GraphicsPath();
            path.AddLine(new Point(10, 10), new Point(400, 10));
            path.AddLine(new Point(400, 10), new Point(400, 250));
            path.AddLine(new Point(400, 250), new Point(10, 250));
            PathGradientBrush pathBrush = new PathGradientBrush(path);
            Color centerColor = Color.Red;
            Color[] surroundColors =
            { Color.Yellow, Color.Green, Color.Blue, Color.Cyan };

            pathBrush.CenterColor = centerColor;
            pathBrush.SurroundColors = surroundColors;

            g.FillPath(pathBrush, path);
        }
    }
}
```



لاحظ ما يحدث عند زيادة تركيز المركز:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            GraphicsPath path = new GraphicsPath();
            path.AddEllipse(0, 0, 200, 100);

            PathGradientBrush pthGrBrush = new PathGradientBrush(path);

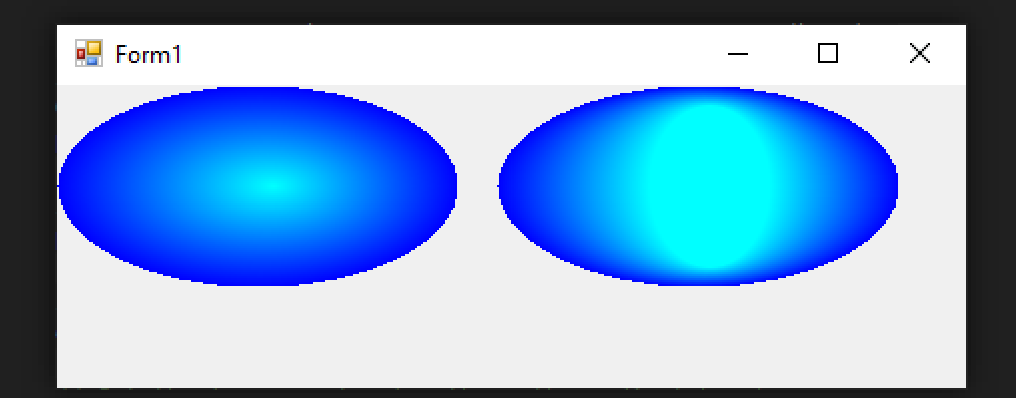
            Color[] color = { Color.Blue };
            pthGrBrush.SurroundColors = color;

            pthGrBrush.CenterColor = Color.Aqua;

            e.Graphics.FillPath(pthGrBrush, path);

            pthGrBrush.FocusScales = new PointF(0.3f, 0.8f);

            e.Graphics.TranslateTransform(220.0f, 0.0f);
            e.Graphics.FillPath(pthGrBrush, path);
        }
    }
}
```



مثال آخر مع ضبط المركز:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

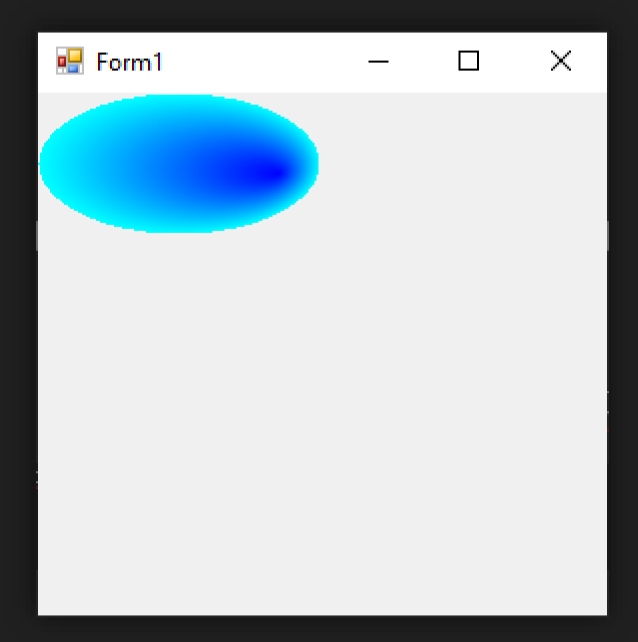
namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            GraphicsPath path = new GraphicsPath();
            path.AddEllipse(0, 0, 140, 70);

            PathGradientBrush pthGrBrush = new PathGradientBrush(path);
            pthGrBrush.CenterPoint = new PointF(120, 40);
            pthGrBrush.CenterColor = Color.FromArgb(255, 0, 0, 255);

            Color[] colors = { Color.FromArgb(255, 0, 255, 255) };
            pthGrBrush.SurroundColors = colors;

            e.Graphics.FillEllipse(pthGrBrush, 0, 0, 140, 70);
        }
    }
}
```



لاحظ النقطة (120, 40)!

وهذا مثال آخر:

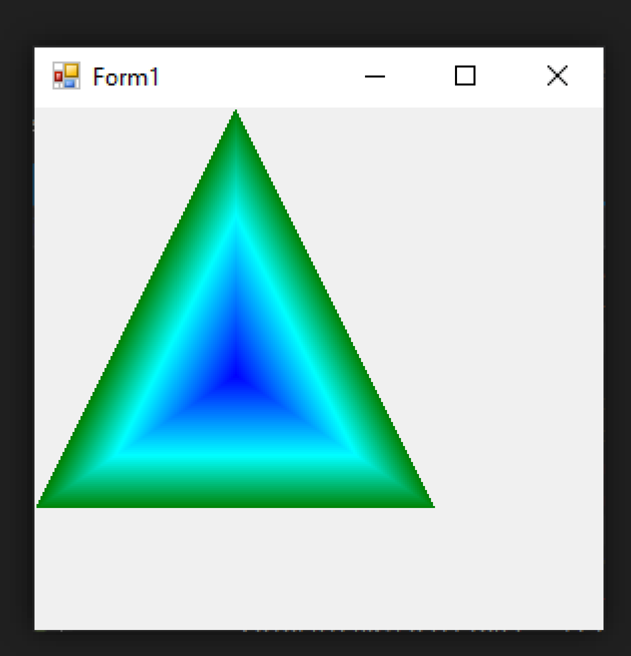


```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Point[] points = {
                new Point(100, 0),
                new Point(200, 200),
                new Point(0, 200)};

            PathGradientBrush pthGrBrush = new PathGradientBrush(points);
            Color[] colors = {
                Color.FromArgb(255, 0, 128, 0),    // dark green
                Color.FromArgb(255, 0, 255, 255),  // aqua
                Color.FromArgb(255, 0, 0, 255)};   // blue
            float[] relativePositions = { 0f, 0.4f, 1.0f};
            ColorBlend colorBlend = new ColorBlend();
            colorBlend.Colors = colors;
            colorBlend.Positions = relativePositions;
            pthGrBrush.InterpolationColors = colorBlend;
            e.Graphics.FillRectangle(pthGrBrush, 0, 0, 200, 200);
        }
    }
}
```



لاحظ هذا:



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            Font f = new Font(new FontFamily("Times New Roman"), 10);
            Brush fb = new SolidBrush(Color.Black);
            GraphicsPath gp;
            PathGradientBrush pGB;
            Rectangle rec;
            Color cR = Color.Red, cW = Color.White, cY = Color.Yellow;
            int w = 100, h = 70;

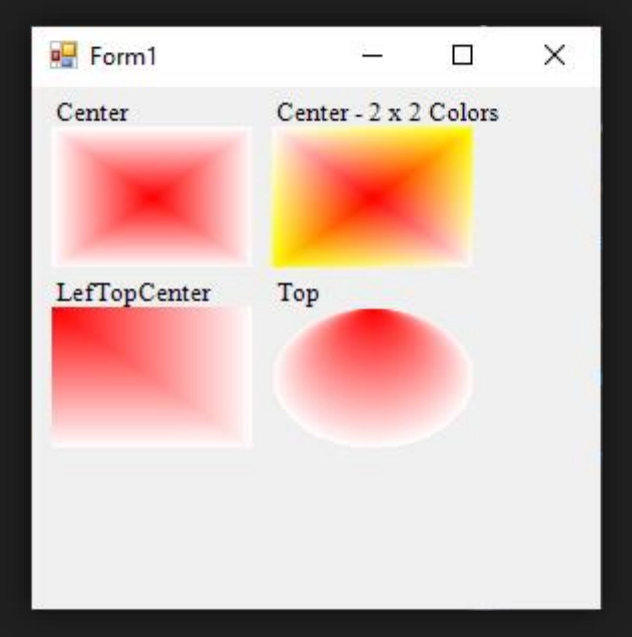
            g.DrawString("Center", f, fb, 10, 5);
            gp = new GraphicsPath();
            rec = new Rectangle(10, 20, w, h);
            gp.AddRectangle(rec);
            pGB = new PathGradientBrush(gp);
            pGB.CenterPoint = new Point(10 + w / 2, 20 + h / 2);
            pGB.CenterColor = cR;
            pGB.SurroundColors = new Color[1] { cW };
            g.FillRectangle(pGB, rec);

            g.DrawString("Center - 2 x 2 Colors", f, fb, w + 20, 5);
            gp = new GraphicsPath();
            rec = new Rectangle(20 + w, 20, w, h);
            gp.AddRectangle(rec);
            pGB = new PathGradientBrush(gp);
            pGB.CenterPoint = new Point(w + 20 + w / 2, 20 + h / 2);
            pGB.CenterColor = cR;
            pGB.SurroundColors = new Color[4] { cW, cY, cW, cY };
            g.FillRectangle(pGB, rec);

            g.DrawString("LefTopCenter", f, fb, 10, h + 25);
            gp = new GraphicsPath();
            rec = new Rectangle(10, h + 40, w, h);
            gp.AddRectangle(rec);
            pGB = new PathGradientBrush(gp);
            pGB.CenterPoint = new Point(10, h + 40);
            pGB.CenterColor = cR;
            pGB.SurroundColors = new Color[1] { cW };
            g.FillRectangle(pGB, rec);
```

```
            g.DrawString("Top", f, fb, w + 20, h + 25);
            gp = new GraphicsPath();
            rec = new Rectangle(w + 20, h + 40, w, h);
            gp.AddEllipse(rec);
            pGB = new PathGradientBrush(gp);
            pGB.CenterPoint = new Point(w + 20 + w / 2, h + 40);
            pGB.CenterColor = cR;
            pGB.SurroundColors = new Color[1] { cW };
            g.FillRectangle(pGB, rec);

            g.Dispose();
            fb.Dispose();
        }
    }
}
```



تهشير الأشكال



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

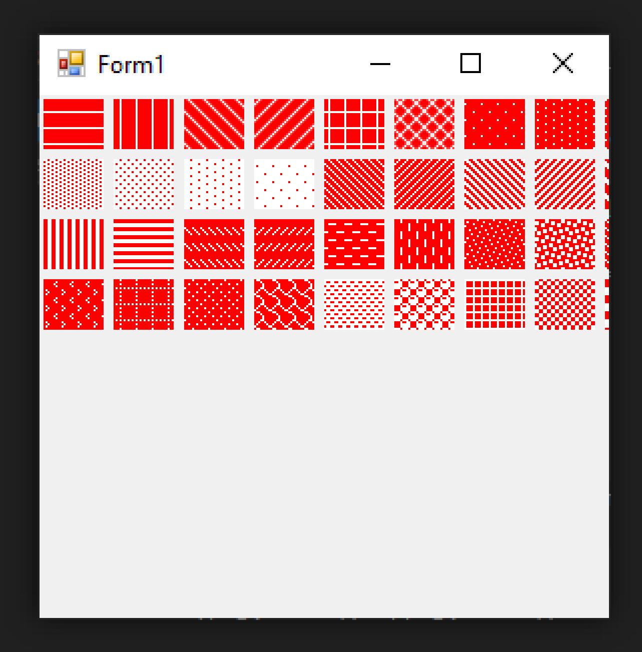
namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            Font f = new Font(new FontFamily("Times New Roman"), 10);
            Brush fb = new SolidBrush(Color.Black);
            Color cb = Color.Red, cf = Color.White;

            int wi = 30, hi = 25, rectNb = 14;
            int x, y;
            HatchBrush hb = null;

            for (int i = 0; i < 53; i++)
            {
                x = (int)(i % rectNb);
                y = (int)(i / rectNb);
                hb = new HatchBrush((HatchStyle)i, cf, cb);
                g.FillRectangle(hb, 2 + x * (5 + wi), 2 + y * (5 + hi), wi, hi);
            }

            fb.Dispose(); hb.Dispose(); g.Dispose();
        }
    }
}
```



تلبيس الرسومات بصورة bmp



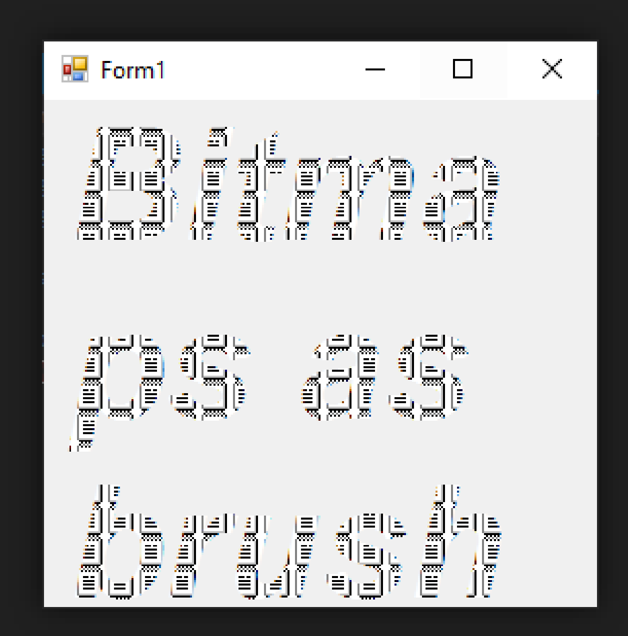
```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        private Brush brush;
        public Form1()
        {
            InitializeComponent();
            ResizeRedraw = true;
            try
            {
                Image img = new Bitmap("txt.bmp");
                brush = new TextureBrush(img);
            }
            catch (Exception e)
            { MessageBox.Show(e.Message); }
        }

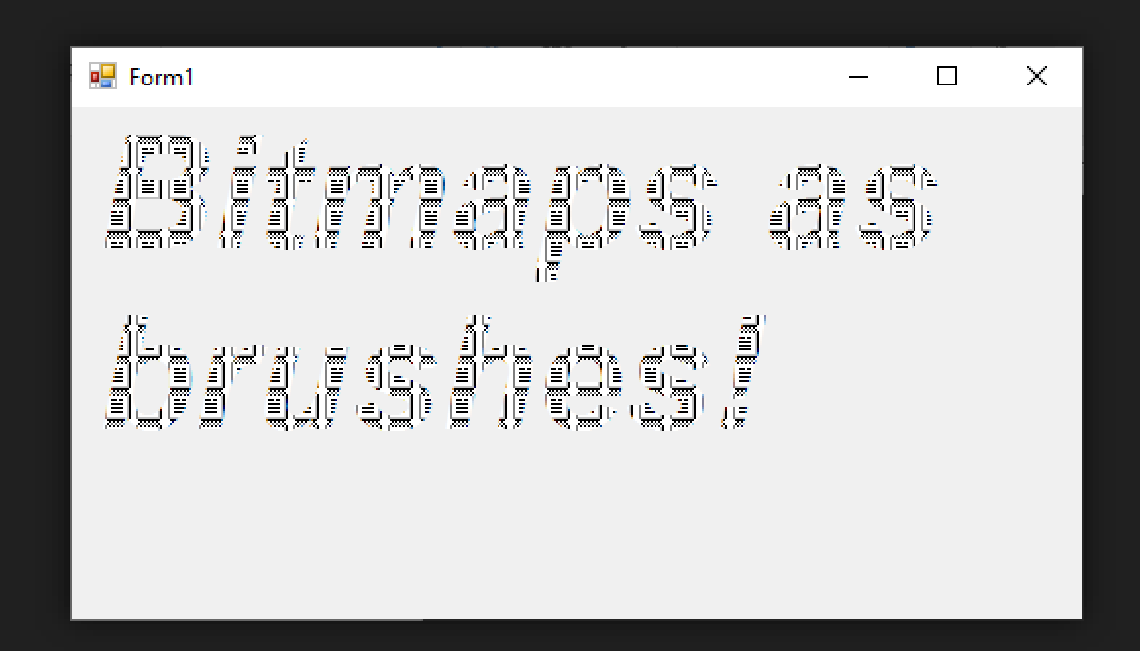
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            Rectangle r = ClientRectangle;

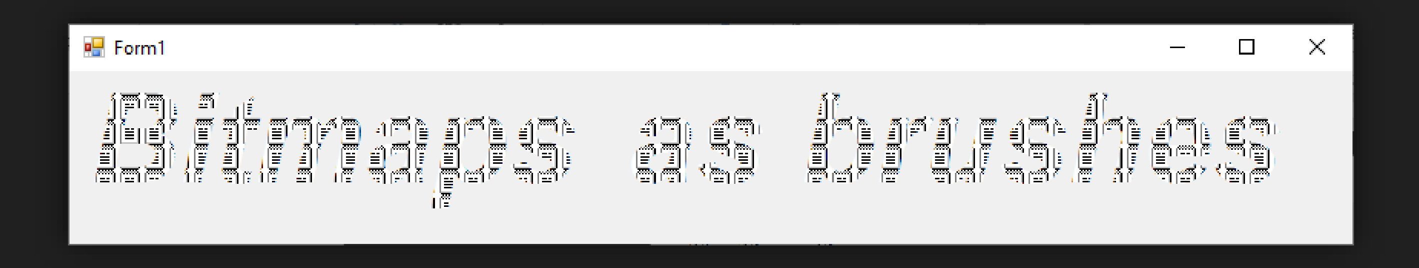
            g.DrawString("Bitmaps as brushes!",
              new Font("Arial", 60,
              FontStyle.Bold | FontStyle.Italic),
              brush,
              r);
        }
    }
}
```

الصورة المستخدمة: .



غير حجم النافذة:





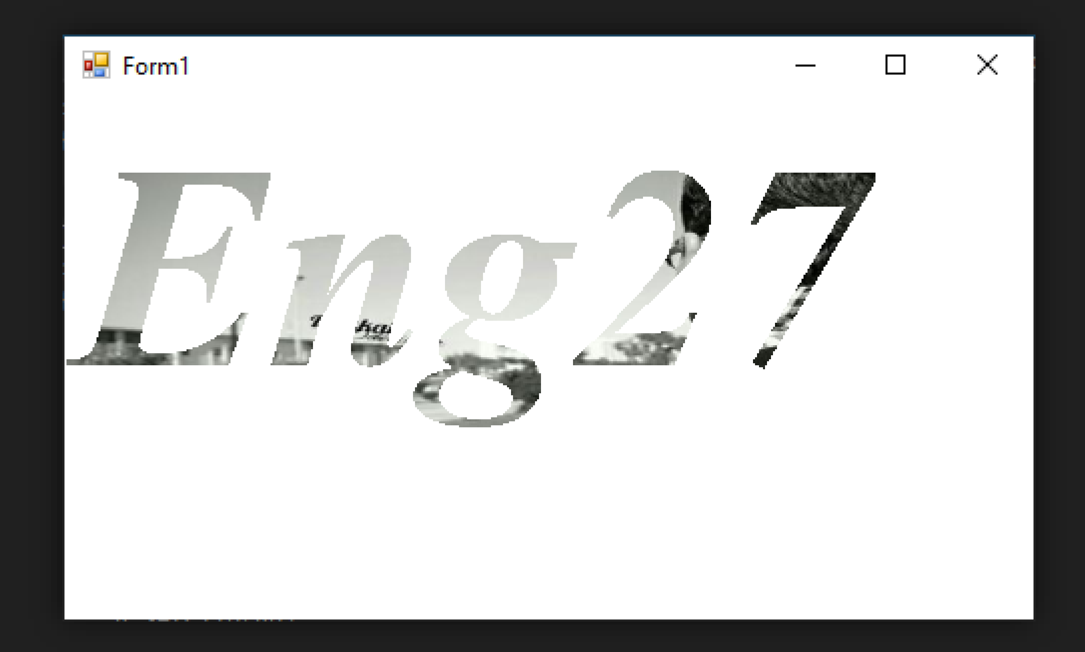
لاحظ هذا المثال:



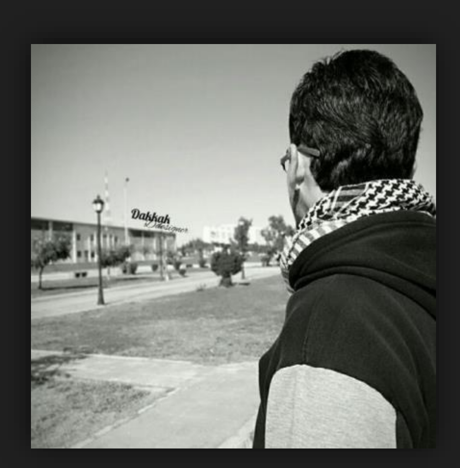
```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            Width = 500;
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.FillRectangle(Brushes.White, this.ClientRectangle);
            GraphicsPath gp = new GraphicsPath();
            gp.AddString("Eng27",
                new FontFamily("Times New Roman"),
                (int)(FontStyle.Bold | FontStyle.Italic),
                144,
                new Point(5, 5),
                StringFormat.GenericTypographic);
            g.SetClip(gp);
            g.DrawImage(new Bitmap("photo.jpg"), this.ClientRectangle);
            gp.Dispose();
        }
    }
}
```



حيث تم استخدام هذه الصورة:



التعامل مع الرسومات باستخدام الفأرة

أول مثال تناولناه في هذا الفصل كان عن الرسم باستخدام الفأرة، أنصحك بالعودة إليه، ثم استئناف قراءة هذه الفقرة.

رسم خط عند الاستمرار بالنقر على منطقة الرسم



```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {

        bool isDrawing;
        Point lastPoint;

        public Form1()
        {
            InitializeComponent();
        }

        protected override void OnMouseDown(MouseEventArgs mea)
        {
            if (mea.Button != MouseButtons.Left)
                return;

            lastPoint = new Point(mea.X, mea.Y);
            isDrawing = true;
        }
```

```
        protected override void OnMouseMove(MouseEventArgs mea)
        {
            if (!isDrawing)
                return;

            Point newPoint = new Point(mea.X, mea.Y);

            Graphics g = CreateGraphics();
            int width = 2;
            g.DrawLine(new Pen(ForeColor, width), lastPoint, newPoint);
            g.Dispose();

            lastPoint = newPoint;
        }

        protected override void OnMouseUp(MouseEventArgs mea)
        {
            isDrawing = false;
        }
    }
}
```

تحديث الرسم مع تغيير حجم منطقة الرسم

لا بد أنك قد لاحظت أن منطقة الرسم لا يتم تحديثها مع تغير حجم منطقة الرسم، خصوصًا عند التعامل مع الرسومات من خلال الفأرة، كالفقرة السابقة. المثال التالي يرسم هوامش للنافذة مع تحديث هذه الهوامش بتغير حجم النافذة:



```
using System.Drawing;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {

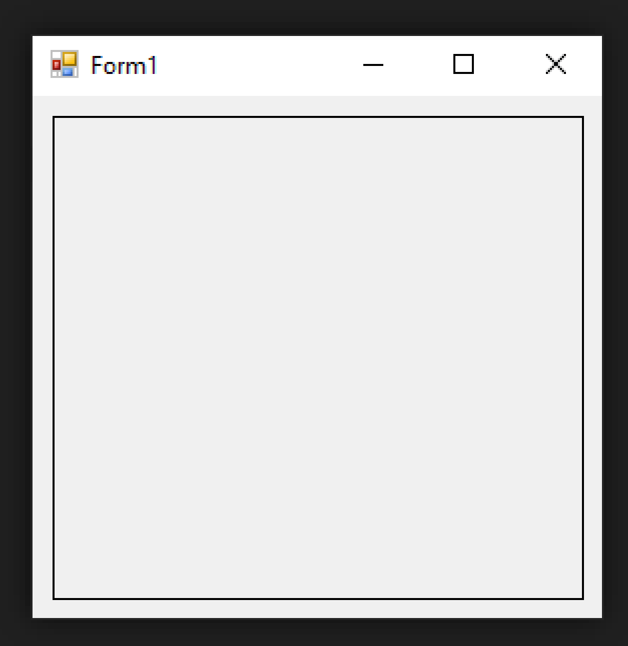
        public Form1()
        {
            InitializeComponent();
            ResizeRedraw = true;
        }

        protected override void OnPaint(PaintEventArgs pea)
        {
            DoPage(pea.Graphics, ForeColor, ClientSize.Width, ClientSize.Height);
        }

        protected void DoPage(Graphics grfx, Color clr, int width, int height)
        {
            Point[] points = {new Point(10 , 10),
                        new Point(width - 10, 10),
                        new Point(width - 10, height - 10),
```

```
                    new Point( 10, height - 10),
                    new Point(10, 10)};

            grfx.DrawLines(new Pen(clr), points);
        }
    }
}
```





لاحظ أنه في هذه الحالة يتم مسح الرسومات القديمة عند حدوث الحدث Paint ورسمها من جديد بالأبعاد الجديدة، على عكس الأمثلة السابقة في هذا الفصل، والتي لم يكن فيها تحديث منطقة الرسومات مدعومًا.

رسم التوابع الرياضية

عند رسم التوابع الرياضية فإنك بحاجة لرسم أمور أخرى كالمحاور الإحداثية وغيرها، ولو أن التابع الرياضي هو الشيء الأساسي في منطقة الرسم، إلا أن هذه الأمور الأخرى هي أيضًا أشياء مفصلية من خلالها يمكنك قراءة وتحليل الخطوط البيانية.

ضبط منطقة الرسم

إذا كانت لديك تجربة مع أداة المخططات Chart فإن ما سنقوم به مألوف لديك.

المتغيرات التالية ستضبط منطقة الرسم، وهي موجزة ومختصرة (هناك الكثير من الأمور تحتاج متغيرات لضبطها):



```
int chart_margin = 50;
int chart_size = 300;
int grid_size = 50;
float grid_lines_size = 0.2f;
int axis_margin = 10;
Color curve_color = Color.Blue;
int curve_size = 2;

int legend_height = 50;
int legend_width = 100;
int legent_margin = 10;

double xstart = -5, xend = 5, dx = 0.1;

Color intersection_points_color = Color.Red;
Color intersection_points_bordercolor = Color.Black;
int intersection_points_size = 6;
bool draw_intersection_points = true;
```

بعض هذه المتغيرات يمكن إظهارها كخيارات ضمن النافذة، ليختار المستخدم ما يراه مناسبًا في حالته، كإحداثيات البداية xstart وإحداثيات النهاية xend ولون المنحني البياني وسمكه وغيرها. كما يمكن حسابها تلقائيًّا.

في كل مرة يُرسم فيها خط بياني يجب استدعاء الطريقة SettingUpGraphics وإسناد القيمة المرجَعة منها إلى متغير من النوع Graphics:



```
Graphics SettingUpGraphics()
{
    Graphics g = this.CreateGraphics();
    g.Clear(this.BackColor);
    g.TranslateTransform(chart_margin, chart_margin, MatrixOrder.Append);
    g.SmoothingMode = SmoothingMode.AntiAlias;
    ClipGraphics(g);
    return g;
}

void ClipGraphics(Graphics g)
{
    g.SetClip(new Rectangle(0, 0, chart_size, chart_size));
}
```

الطريقة ClipGraphics تقتطع منطقة الرسم حتى لا يُرسم شيء خارجها.

رسم المحاور الإحداثية

الإجراء CreateAxes يأخذ كائنًا من النوع Graphics يمثل منطقة الرسم:

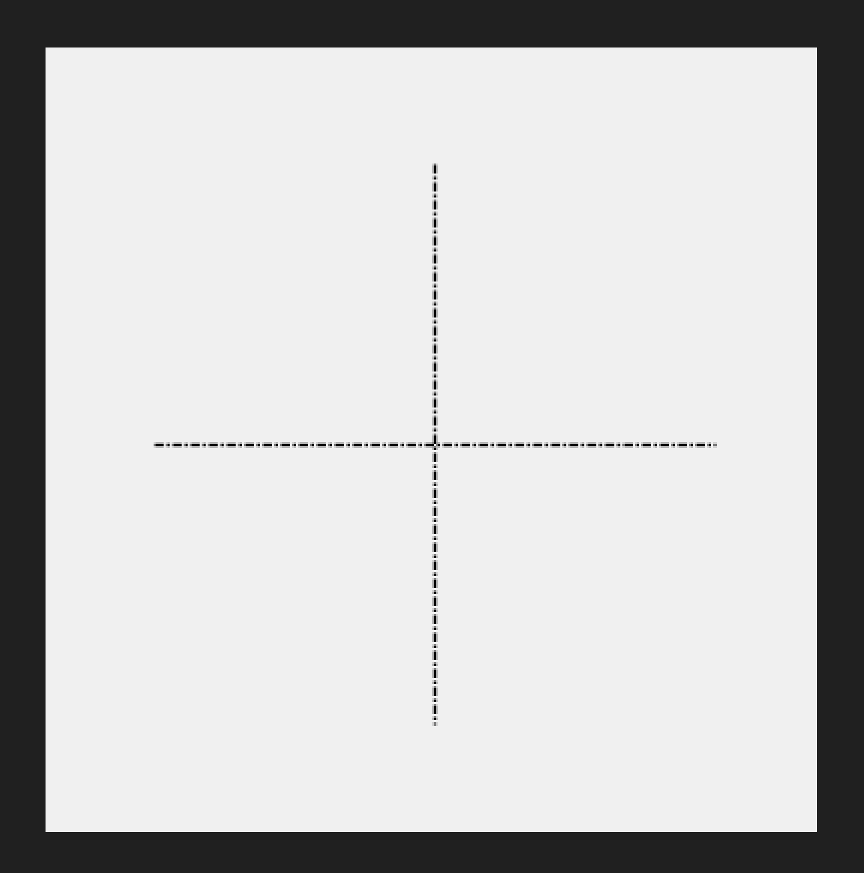


```
void CreateAxes(Graphics g)
{
    Point p1, p2;

    Pen pen = new Pen(Color.Black,1.5f);
    pen.DashStyle = DashStyle.DashDot;

    p1 = new Point(chart_size / 2, axis_margin);
    p2 = new Point(chart_size / 2, chart_size - axis_margin);
    g.DrawLine(pen, p1, p2);

    p1 = new Point(10, chart_size / 2);
    p2 = new Point(chart_size-10, chart_size / 2);
    g.DrawLine(pen, p1, p2);
}
```

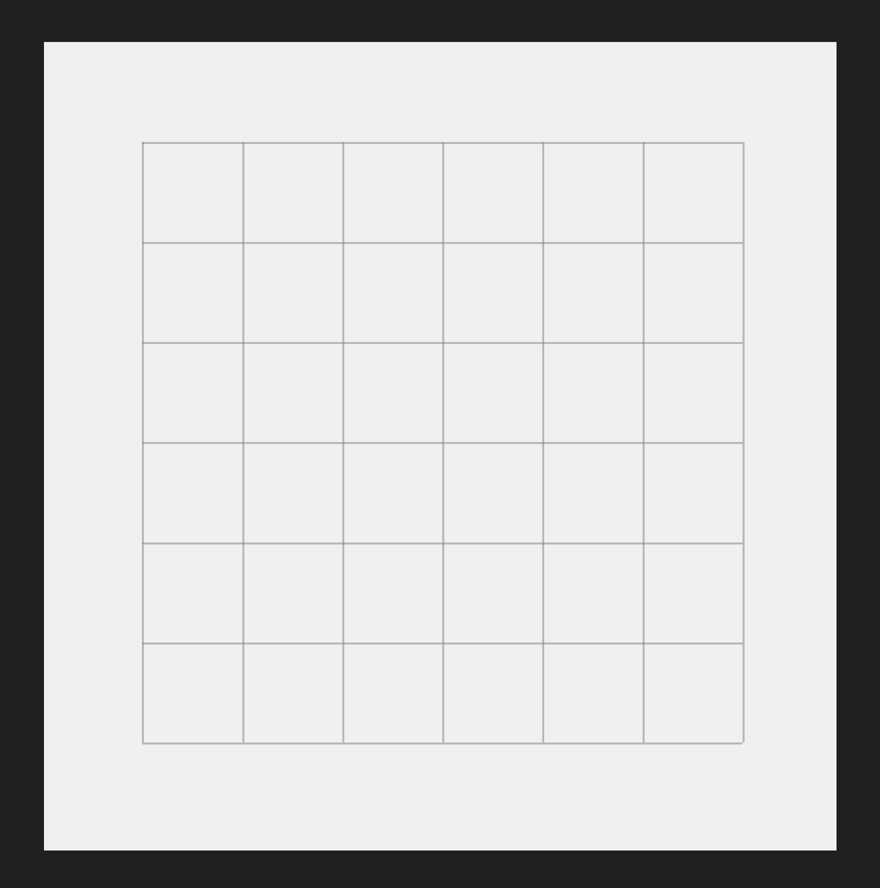


رسم شبكة Grid

تساعد الشبكة Grid في فهم المخططات البيانية أكثر، يمكنك تطوير الطريقة التالية لتعطي المستخدم الخيار لعرض الشبكة بالشكل الذي يريد، بما في ذلك حجم الخط ولونه، وهل يرغب بعرض الخطوط الأفقية أو الشاقولية أو كليهما أو عدم عرض الشبكة.



```
void CreateGrid(Graphics g)
{
    Point p1, p2;
    Pen pen = new Pen(Color.Gray, grid_lines_size);
    pen.DashStyle = DashStyle.Dot;
    for (int i = 0; i <= chart_size; i += grid_size)
    {
        p1 = new Point(0, i);
        p2 = new Point(chart_size, i);
        g.DrawLine(pen, p1, p2);
    }
    for (int i = 0; i <= chart_size; i += grid_size)
    {
        p1 = new Point(i, 0);
        p2 = new Point(i, chart_size);
        g.DrawLine(pen, p1, p2);
    }
}
```



رسم عنوان تفصيلي Legend

عند وجود منحنيات كثيرة في المخططات تستخدم ما تسمى باللافتات أو العناوين التفصيلية لعنونة هذه المنحنيات والمكونات المختلفة، وهي عبارة عن مستطيل يرسم في إحدى زوايا المخطط. لن ندخل في تفاصيل اللافتات، وإنما سنركز على رسم المستطيل الذي يمثلها فقط:



```
void CreateLegend()
{
    Graphics g = this.CreateGraphics();
    int legend_x = chart_margin + chart_size - legend_width - legent_margin;
    int legend_y = chart_margin + chart_size - legend_height - legent_margin;
    g.TranslateTransform(legend_x, legend_y, MatrixOrder.Append);
    Rectangle rect = new Rectangle(0, 0, legend_width, legend_height);
    g.FillRectangle(Brushes.White, rect);
    g.DrawRectangle(Pens.Black, rect);

    //..
}
```

آلية رسم المنحنيات

تتكون المنحنيات البيانية من عدد لا نهائي من النقاط، حتى تلك التي يوجد ما يمثلها من معادلات أو دساتير أو قوانين، كالمستقيمات والدوائر والقطوع وغيرها، فأسهل سبيل لرسم هذه الخطوط من خلال الحاسوب هو رسم نقاط تنتمي إليها، وعلى اعتبار أعمارنا محدودة، فلا بد من رسم عدد محدود من النقاط.

عدد نقاط أي منحني تود رسمه يتعلق بخطوته، وأعني بخطوة المنحني الفرق بين إحداثيي نقطتين متتاليتين منه، بفرض أن جميع نقاط المنحني تبعد عن بعضها وفق المحور x قيمة واحدة. المعادلة التالية تعطيك عدد نقاط المنحني بدلالة خطوته dx وبدايته x_{start} ونهايته x_{end}:

$$n = \frac{x_{end} - x_{start}}{dx}$$

بعد تحديد عدد النقاط المكونة للمنحني، يمكنك إنشاء مصفوفة من النقاط لتخزين نقاط المنحني، هذا الأسلوب – تخزين بيانات المنحني في مصفوفة – هو أسهل وأبسط وأفضل أسلوب يمكن سلوكه، ويكاد يكون السبيل الوحيد عند التعامل مع المنحنيات.

أما عن الكيفية التي سيتم فيها الرسم، فهو بتعويض قيمة المتغير x في التابع y، حيث x يتغير من x_{start} إلى x_{end}، بفارق dx (أو step) بين قيمة ما لهذا المتغير وما تليها. هذا يعني أن قيمة المتغير x هي قيمة من متتالية حسابية، يمكنك معرفة أي عنصر منها بمعرفة ترتيب هذا العنصر.

يمكن معرفة قيمة التابع y في كل نقطة من نقاط المنحني بتعويض قيم x في معادلة التابع، وبعدها – بعد إيجاد جميع نقاط المنحني – يمكنك رسم منحني من خلال الطريقة DrawCurve.

الإجراء التالي يجمع كل ما سبق من طرق ناقشناها، وفيه يتم رسم التوابع الرياضية، كما سنرى في الفقرات القادمة.



```
private void btnDraw_Click(object sender, EventArgs e)
{
    Graphics g = SettingUpGraphics();
    CreateGrid(g);
    CreateAxes(g);

    // مقياس رسم المخطط
    double xscale = grid_size, yscale = -grid_size;
    int w = chart_size, h = chart_size;
    Point origin = new Point(w / 2, h / 2);

    // حساب طول المصفوفة – عدد نقاط المنحني
    int n = (int)((xend - xstart) / dx) + 1;

    double[] x = new double[n];
    double[] y = new double[n];
    Point[] p = new Point[n];
    List<Point> intersect_points = new List<Point>();

    for (int i = 0; i < n; i++)
    {
        x[i] = xstart + (i * dx);
        y[i] = ...; // معادلة التابع
        p[i].X = origin.X + (int)(x[i] * xscale);
        p[i].Y = origin.Y + (int)(y[i] * yscale);

        if (x[i] == 0 || y[i] == 0)
        {
            intersect_points.Add(p[i]);
        }
    }

    Pen pen = new Pen(curve_color, curve_size);
    g.DrawCurve(pen, p);

    if (draw_intersection_points)
        DrawIntersectionPoints(intersect_points.ToArray(), g);

    CreateLegend();
    g.DrawRectangle
       (Pens.Black, new Rectangle(0, 0, chart_size - 1, chart_size - 1));
}
```

في البداية يتم تجهيز منطقة الرسم (مسح النافذة، وضبط حجم منطقة الرسم وغيرها وإنشاء الشبكة، وبعدها المحاور الإحداثية.

ثم يتم ضبط مقياس رسم المحاور الإحداثية للمخطط، وهنا ضع في ذهنك نقطتين:

- لجعل مقياس الرسم حقيقي، اضبطه على القيمة 37.795، للتحويل لواحدة سم. يمكنك القيام بذلك للشبكة أيضًا.
- محاور النوافذ وأسطح الرسم في الحاسوب ليست نفسها المحاور x وy التي نتعامل معها رياضيًّا، مبدأ الإحداثيات في مناطق الرسم هو الزاوية اليسرى العليا، المحور الأفقي نفسه محور x أما المحور الشاقولي فعكسه، لذلك فقد عكسناه بضربه بالمقياس 1-. كما أنه لا معنى للأبعاد السالبة في أسطح الرسم في الحاسوب، ولو أنها موجودة واستخدامها ممكن، إلا أن كل ما تتعامل معه يجب أن يكون في القسم الموجب من المحاور الإحداثية (بكلام رياضي: بالربع الأول).

عدد نقاط المنحني ثابتًا بالنسبة للشروط التي فرضناها (الخطوة وغيرها)، لكن عدد نقاط التقاطع متغير لنفس الشروط؛ لذلك فإننا لا نعرف عدد نقاط التقاطع، وبالتالي يجب تخزينها ضمن لائحة List وليس مصفوفة Array. عند التعامل مع المصفوفات فإننا نستخدم الحلقات، ومنها الحلقة for وهي الأكثر شيوعًا. إن التابع الذي نرسمه يبدأ من x_{start} وينتهي بـ x_{end}، فإذا لم ترغب بتقييد المنحني فاجعل الإحداثي يبدأ من 0 وليس من x_{start} وينتهي ليس عندما لا يتحقق الشرط $i < n$ في حلقة التكرار وإنما عندما تصبح x – بعد تحويلها إلى مقياس النقطة Pixel – مساوية أو أكبر من عرض منطقة الرسم. وأما بالنسبة لنقاط التقاطع (الشرط الذي يفترض أن النقطة التي أحد إحداثياتها معدومة)، فسنأتي عليها في الفقرة التالية.

وأخيرًا بعد الحصول على جميع نقاط المنحني، ونقاط التقاطع، يمكن رسمه من خلال الطريقة DrawCurve، وإذا فعّل المستخدم خيار رسم نقاط التقاطع فإنها ترسم أيضًا. ثم يرسم العنوان التفصيلي Legend ثم مستطيل يحيط بمنقطة الرسم.

الجدير بالذكر أن ترتيب الأوامر يحدد مَن سيظهر بالكامل على منطقة الرسم، وبكلام مصممي الغرافيك: الأمر الأخير طبقته Layer أعلى. فلو استدعيت الطريقة DrawLegend أولًا لظهر المنحني البياني والشبكة – وسائر الأشياء التي سترسمها – فوق صندوق العنوان التفصيلي.

لجعل بيئة الرسم هذه أكثر فائدة، اجعلها كائنية التوجه، أنشئ طرقًا لرسم الأشكال المعروفة، وطرقًا لرسم الأشكال العامة. سنتناول بعض التوابع المشهورة في الفقرات القادمة، ولكنا لن نتطرق لكيفية تطويرها وجعلها كائنية التوجه، بل سنستخدم نفس كود هذه الفقرة وننسخه في كل مرة نرسم فيها تابعًا جديدًا. وفي نفس الوقت، أعتبر أنك على دراية بأن هذا الأسلوب ليس صحيحًا، وإنما استخدمناه لإيصال الفكرة فقط وليس لتطوير أداة أو فئة ما، فإني آمل ألّا أرى برنامجًا يقوم على هذا الأسلوب، فإنه يؤلم قلبي كما سيؤلم قلب المترجم Compiler.

رسم نقاط التقاطع مع المحاور الإحداثية

لنقاط تقاطع الخطوط البيانية للتوابع الرياضية مع المحاور الإحداثية أهمية كبيرة من الناحية الرياضية، وتوجد هذه النقاط – رياضيًا – عندما تحقق نقطة ما من التابع معادلة أحد المحاور الإحداثية، وبكلام آخر: عندما تكون معادلة التابع مساوية لمعادلة أحد المحاور الإحداثية. لا عليك إذا لم تفهم الكلمات السابقة فهذا متوقع، كما أن الطريقة التالية المسؤولة عن رسم نقاط التقاطع لا تحوي معادلات رياضية ولا أمور توجع الرأس، فانسخ الكود فقط وبلا فضايح 🙂.



```
void DrawIntersectionPoints(Point[] points, Graphics g)
{
    SolidBrush brush = new SolidBrush(intersection_points_color);
    Pen pen = new Pen(intersection_points_bordercolor);
    int p_margin = intersection_points_size / 2;
    int p_size = intersection_points_size;
    foreach (Point p in points)
    {
        // اترك للمستخدم الخيار ماذا يريد أن تكون نقاط التقاطع
        // دوائر أو مربعات أو غيرها

        g.FillEllipse
           (brush, new Rectangle(p.X - p_margin, p.Y - p_margin, p_size, p_size));
        g.DrawEllipse
           (pen, new Rectangle(p.X - p_margin, p.Y - p_margin, p_size, p_size));
    }
}
```

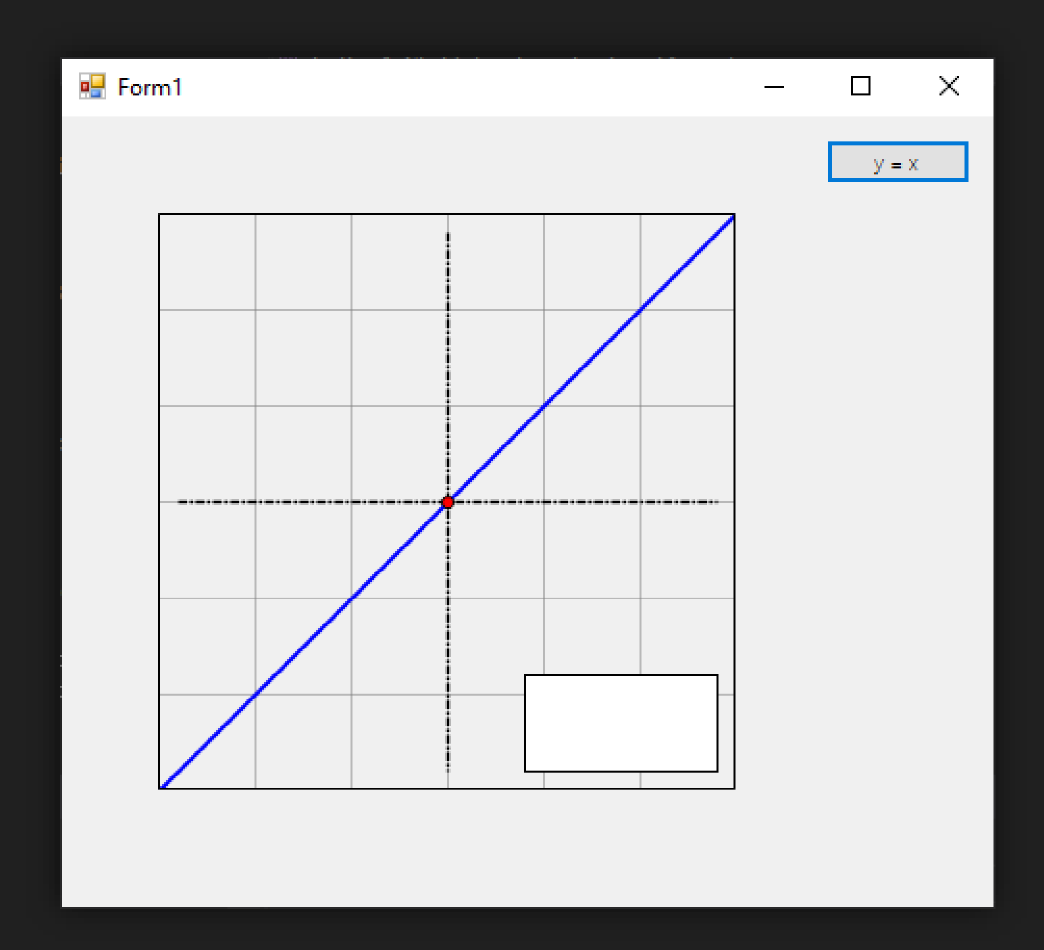
الطريقة DrawIntersectionPoints تأخذ مصفوفة من النقاط وكائنًا يمثل كائن الرسم، تفترض هذه الطريقة أن النقاط التي ستأخذها من خلال الوسيط points هي نقاط تقاطع،

لكنها لا تملك الوعي الكافي لتحديد هل فعلًا هذه النقاط هي نقاط تقاطع لمنحني التابع مع المحاور الإحداثية أو لا.

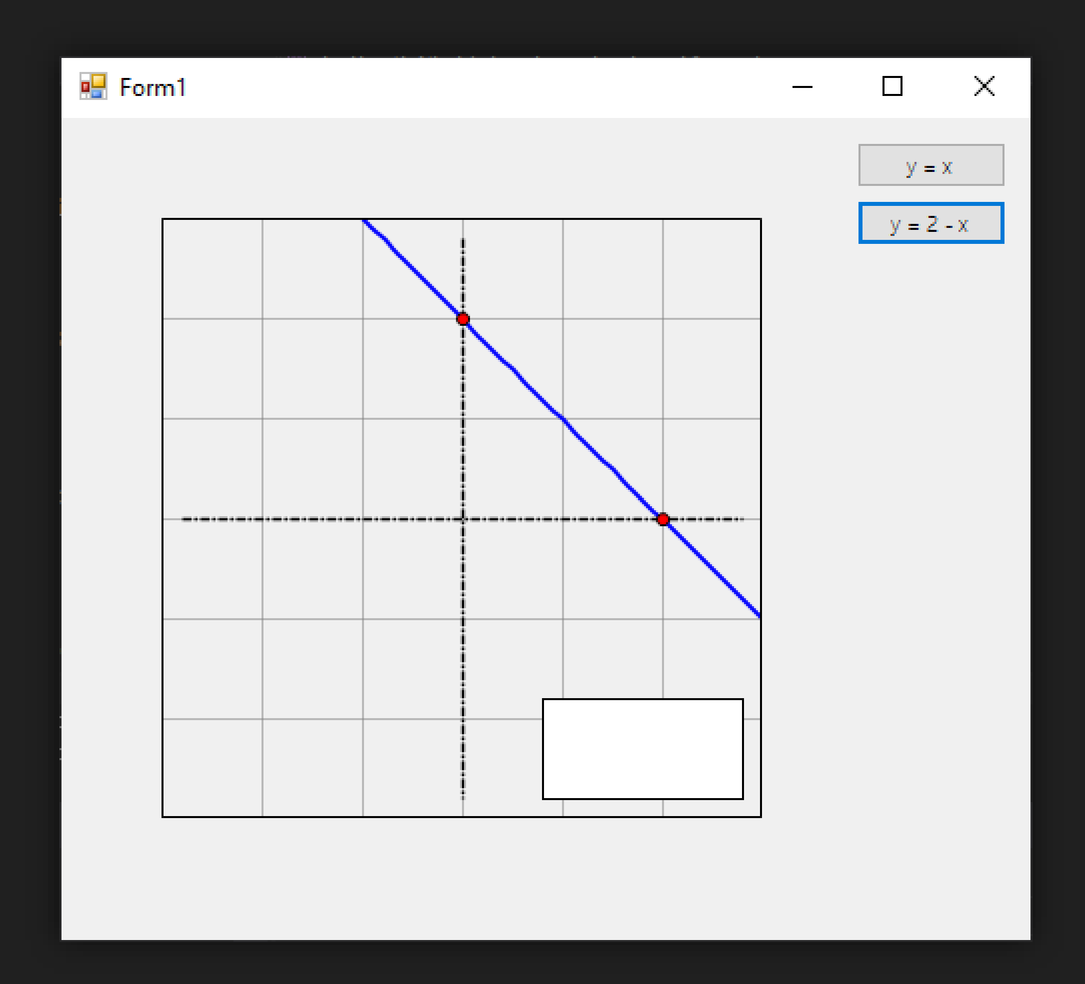
يمكنك استخدام هذه الطريقة في رسم النقاط العادية حتى، أو لرسم نقاط تقاطع المنحنيات البيانية مع بعضها، فطورها وفق حاجتك. (في الواقع فإن عملية التطوير التي تحتاج أن تقوم بها لا تزيد عن تعديل اسم الطريقة وتعديل افتراضك عن ماهية الوسيط points).

رسم الخط البياني لمستقيم

تعطى المستقيمات بمعادلات خطية (معادلات من الدرجة الأولى)، وكمثال: لنرسم المستقيم y = x (وهو مستقيم مار من المبدأ منصّف للربع الأول). أنشئ زرًا واكتب فيه في الحدث Click الكود الموجود في الفقرة **آلية رسم المنحنيات** واجعل معادلة المنحني y[i] = x[i].



وكمثال آخر: المستقيم y = 2 – x هو نفسه المستقيم الذي رسمناه ولكنه معكوس (منصف للربع الثاني وليس الأول) وغير مار من المبدأ (مزاح بمقدار 2 باتجاه المحور y). لذلك اجعل معادلة المنحني y[i] = 2 – x[i].



لاحظ أن المقدار 2 نفسه مقدار خطوة الشبكة، لأننا جعلنا مقياس الرسم نفسه مقياس الشبكة (خزّناه في المتغير gridsize). لاحظ أيضًا عندما تنقر على الزر الأول ثم تنقر على الزر الثاني فإن منطقة الرسم سيتم مسحها ورسمها من جديد، وهذا بفضل الطريقة SettingUpGraphics، التي تحوي أمرًا يقوم بذلك.

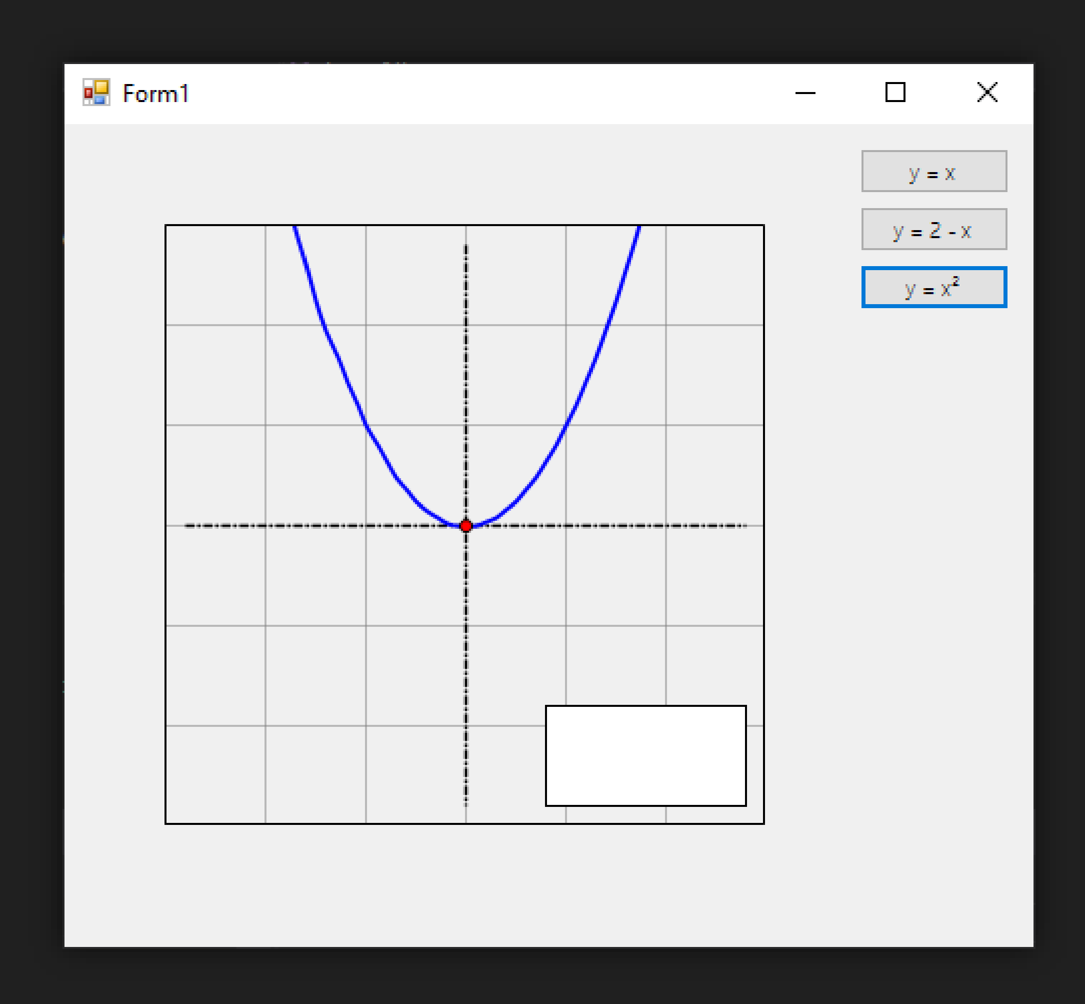
يمكنك تطوير طريقة رسم المستقيم، وذلك بإعادة تعريفها لجعلها تقبل وسطاء لحالات مختلفة، كجعلها تقبل وسيطين: الميل ونقطة من المستقيم، أو نقطتين منه، أو بأسلوب متقدم: تعطي البرنامج معادلة تمثل منحنٍ ما، ونقطة تماس المستقيم معها، فيشتق

هذه المعادلة رياضيًّا ليجد ميل المستقيم، ويعود للطريقة الأولى فيرسم المستقيم بمعرفة ميله ونقطة منه.

رسم الخط البياني لقطع مكافئ

القطوع هي أشكال هندسية مشهورة كالدائرة والقطع الناقص، وهي ليست خطية كالمستقيم (معادلتها ليست من الدرجة الأولى).

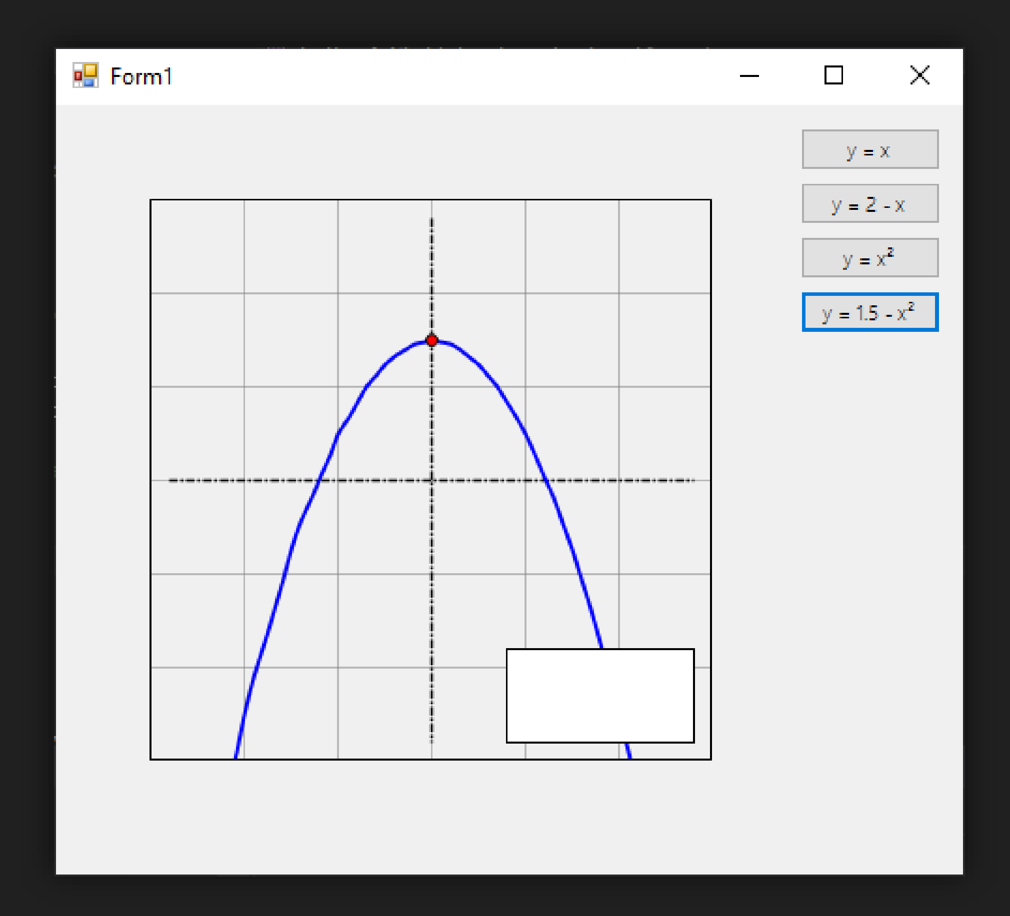
سأتناول القطع المكافئ لشرح رسم المعادلات غير الخطية، ولو أن الدائرة أشهر، إلا أنه أسهل، خصوصًا من ناحية إيجاد علاقة للتابع y بالنسبة للمتغير x. لنأخذ على سبيل المثال المعادلة $y = x^2$، لذلك فأنشئ زرًا جديدًا وانسخ كود الرسم الذي استخدمناه لرسم المستقيمات واجعل معادلة التابع y[i] = Math.Pow(x[i], 2).





إن الخط البياني محدود وليس لا نهائيًا كما قد يخيل إليك، إلا أن منطقة الرسم لا تعرض إلا جزءًا منه بحسب مقياس الرسم. جرب غير بداية ونهاية التابع ولاحظ.

جرب هذا التابع $y = 1.5 - x^2$:



ومن الواضح أن الكود المستخدم قد فشل في تحديد نقاط التقاطع، فمن المفترض أن تكون ثلاث نقاط وليس نقطة، ولتحديد سبب هذا الفشل علينا الانتقال للسطر البرمجي المسؤول عن تحديد نقاط التقاطع، وهو هذا:

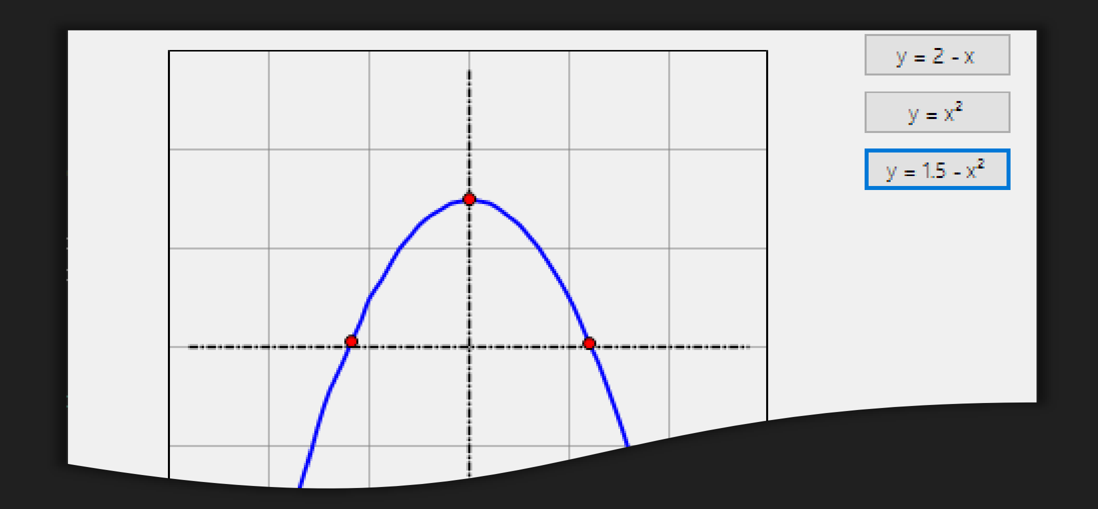
```
if (x[i] == 0 || y[i] == 0)
{
    intersect_points.Add(p[i]);
}
```

لم تكن المشكلة بتحديد نقاط تقاطع الخط البياني مع محور y′y، إذًا فالمشكلة ليست بفواصل النقاط x. بينما لم يتمكن الكود من تحديد نقاط التقاطع مع المحور x′x، وعليه فإن الخلل هو في تراتيب النقاط y.

أما عن سبب عدم كون المشكلة في الفواصل، هو أن قيم فواصل النقاط معلومة مسبقًا، على اعتبارها متتالية حسابية. وأما عن سبب ظهور المشكلة مع التراتيب، فهو أن قيمها لا يمكن التنبؤ بها، صحيح أنه يمكن التنبؤ بقيم تقريبية لها، إلا أن القيم لا يمكن تحديدها بدقة إلا بحسابها من أجل جميع النقاط! فالشرط y[i] == 0 لن يكون محقّقًا إلا عندما تكون y[i] مساوية لـ 0 فعلًا وليست قريبة منها، وهذا لا يمكن التنبؤ به، فقد تكون أقرب قيمة للصفر هي 0.001 مثلًا!

يمكنك حل المشكلة بتطوير الشرط كالتالي:

```
if (Math.Round(x[i], 1) == 0 || (Math.Round(y[i], 1) > -0.1 && Math.Round(y[i], 2) < 0.1))
{
    intersect_points.Add(p[i]);
}
```



لاحظ أن نقاط التقاطع مع المحور الأفقي فعلًا ليست واقعة عليه، أي أن تراتيبها ليست مساوية للصفر تمامًا. يمكنك تطوير طريقة خاصة بإيجاد نقاط التقاطع، وذلك بطرق رياضية تحليلية، وفي هذه الحالة لا حاجة لك لبنية الشرط التي تتحقق من النقاط بعد تعويضها.

وعلى سيرة القطع المكافئ، هل تذكر المعادلة $y = a x^2 + b x + c$؟ هل تذكر حلها؟ x_1 وx_2 والمميز Δ؟ إن هذه المعادلة في الواقع هي معادلة القطع المكافئ، فيمكن مثلًا تطوير طريقة رسم القطع المكافئ بجعلها تأخذ الثوابت a وb وc كوسطاء وحل المعادلة ثم رسمها، أو بجعلها تأخذ جذور المعادلة x_1 وx_2 (وهي نفسها نقاط التقاطع التي فشلت طريقتنا في إيجادها)، وهكذا.

منحني بيزييه

قبل البدء بالمثال، سنذكر نبذة عن المنحني، بالإضافة لما ذكرناه سابقًا..[1]

يُنشأ منحني بيزييه اعتمادًا على مجموعة نقاط، درجة المنحني تُحدَّد من خلال عدد نقاط المنحني، وتساوي عدد نقاط المنحني ناقص واحد.

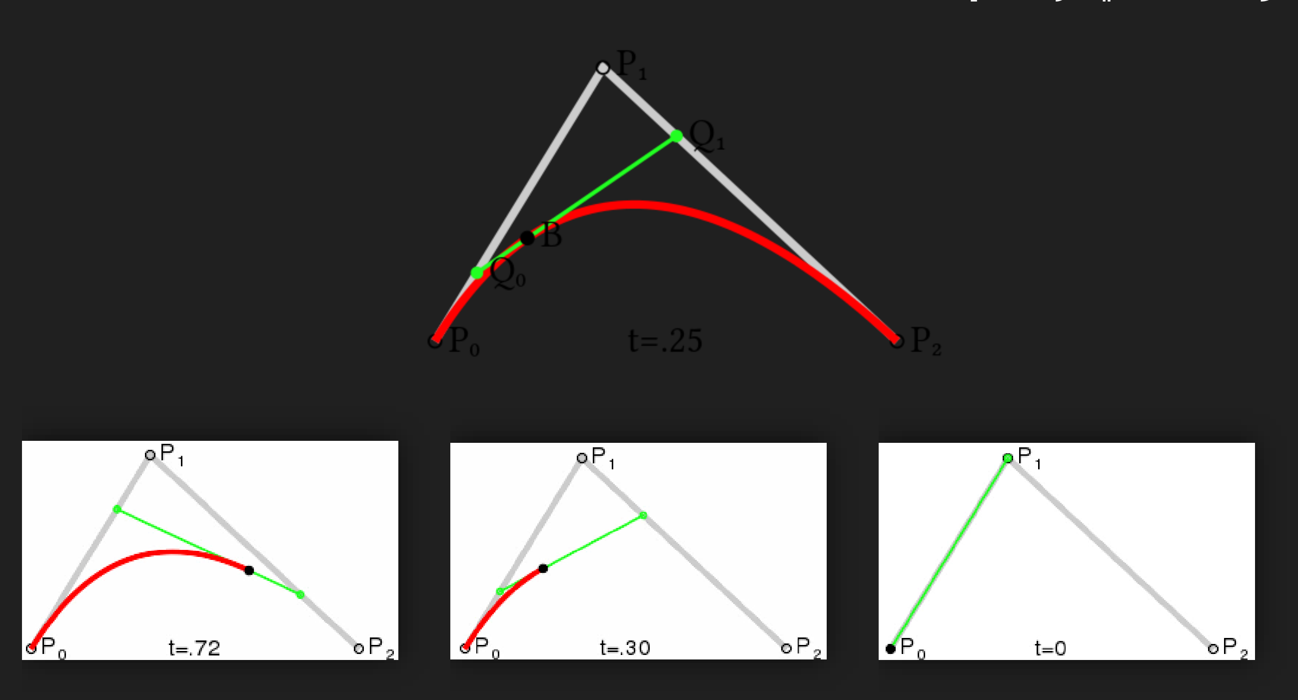
للمنحني تطبيقات رياضية وهندسية كثيرة، وهو من المنحنيات المعقدة المتقدمة؛ لأن المنحنيات الناتجة عنه تتغير بشكل كبير تبعًا للنقاط التي تقيدها. وإذا كان لديك باع طويل مع الرياضيات، وأردت تصميم أشكال هندسية لا توفرها لك المنحنيات التقليدية؛ فتعامل معه مع أمثال هذا المنحني.

يكون المنحني خطيًّا إذا كانت قد شكلته نقطتان، ويكون عندها مستقيمًا:

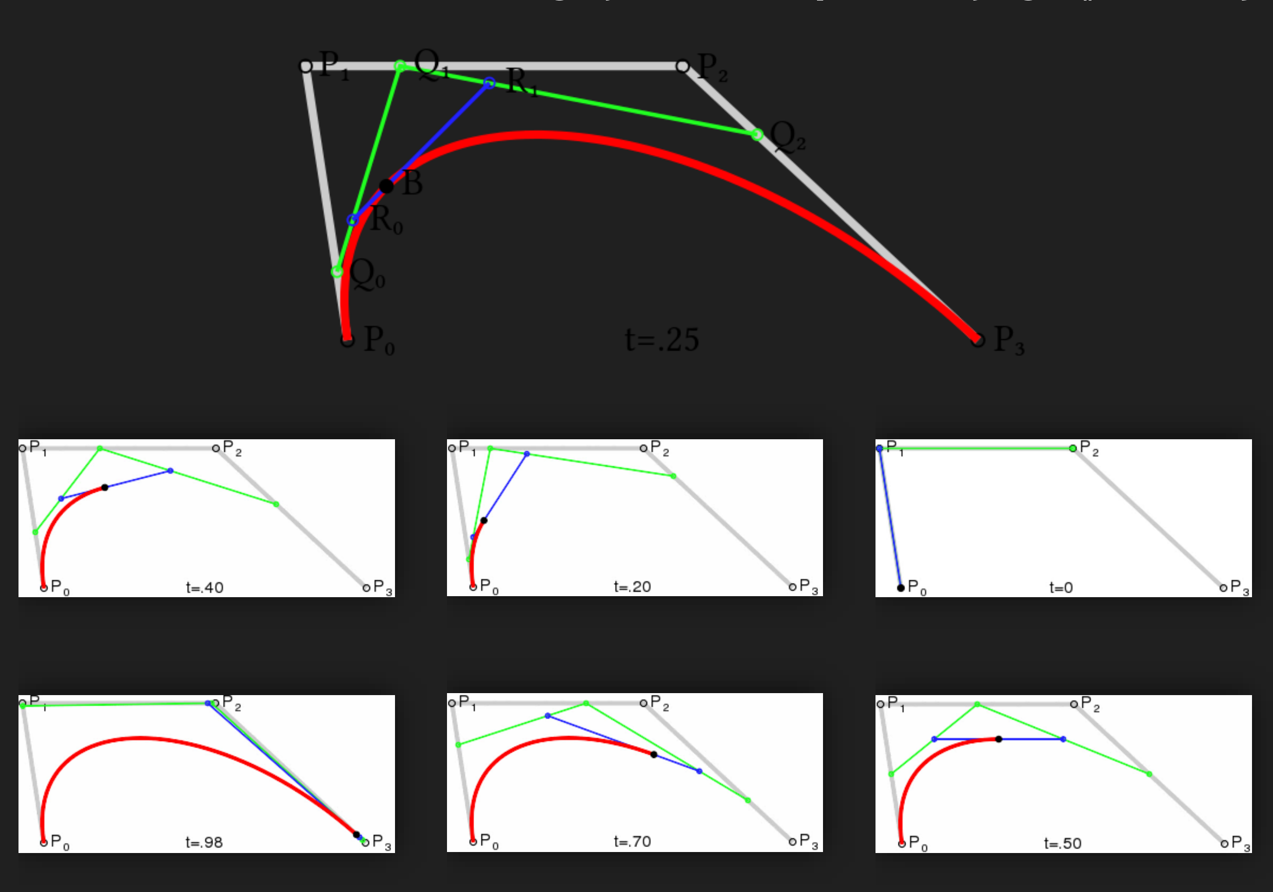


[1] الشرح المذكور في هذه الفقرة من موقع wikiwand، https://www.wikiwand.com/en/B%C3%A9zier_curve.

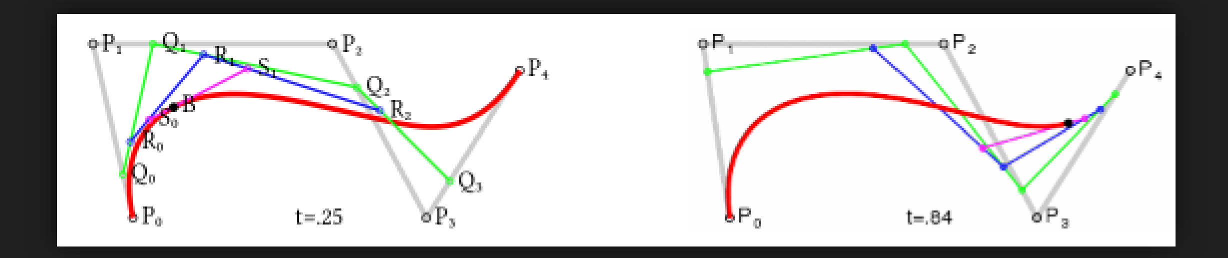
يكون المنحني تربيعيًّا إذا شكلته ثلاث نقاط:



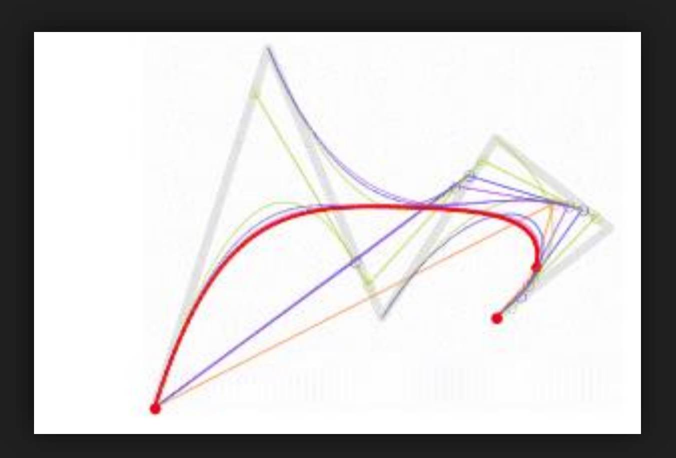
ويكون المنحني من درجات عليا إذا شكلته أكثر من ثلاث نقاط:



ماذا لو خمس نقاط؟



ماذا لو ستة؟



أنشئ نافذة ثانية واضبطها على الوضع WindowState = Maximized. واضبط النافذة الأصلية على لون الخلفية 32, 32, 32 والوضع StartPostition = CenterScreen، وأضف فيها عنوانًا Label بلون خط أبيض وبخط Segoe UI Light, 18pt، واكتب فيها ما تشاء (مثلًا GraphicsTest by Eng27).. ثم استخدم الكود:

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace GraphicsTest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
        int t = 0;

        void Bezier(Control control, int b1, int b2)
        {
            GraphicsPath g2 = new GraphicsPath();
            int n = b1, m = b2;

            g2.AddLine(m,
                0,
                control.Width - m,
                0);
            g2.AddBezier(control.Width - m,
                0,
                control.Width - n,
                0,
                control.Width,
                n,
                control.Width,
                m);

            g2.AddLine(control.Width, m,
                control.Width,
                control.Height - m);
            g2.AddBezier(control.Width,
                control.Height - m,
                control.Width,
                control.Height - n,
                control.Width - n,
                control.Height,
                control.Width - m,
                control.Height);

            g2.AddLine(control.Width - m,
                control.Height,
                m,
                control.Height);
            g2.AddBezier(m,
                control.Height, n,
                control.Height, 0,
                control.Height - n,
                0,
                control.Height - m);

            g2.AddLine(0,
                control.Height - m,
                0,
                m);
            g2.AddBezier(0,
                m,
                0,
                n,
                n,
                0,
                m,
                0);

            control.Region = new Region(g2);
        }
```

```
        private void timer1_Tick(object sender, EventArgs e)
        {
            t += 5;
            if (t == 500)
            {
                t = 0;
                timer1.Enabled = false;
                Form2 f = new Form2();
                f.Show();
                this.Hide();
            }
            Bezier(this, t + 200, t);
        }
    }
}
```

شغّل المشروع ولاحظ النتائج.



الباب الثالث

أدوات المستخدم

UserControl

إذا لم تسعفك الأدوات التي قدمتها لك مايكروسوفت – أو الشركات الأخرى من هنا أو هناك – فيمكنك صنعها بنفسك، ولتصل لتلك المرحلة التي تقوم فيها بإنشاء أدوات لم يقم بصنعها أحد قبلك يجب عليك بدايةً فهمَ ما هو متوفر حاليًّا وهذا ما حاولنا مناقشته وسرده وإيجازه في الباب الأول، ثم الإلمامَ بطرق الرسم وأساليبه وهذا ما تناولناه في الباب الثاني بشيء من التفصيل، ثم التحلي ببعض الذوق الفني لإضفاء مفاهيم الـ UI على تطبيقاتك وهذا ما أوردنا شيئًا منه في أرجاء البابين السابقين، بالإضافة لفهمك للهندسة الاجتماعية لتعطي برنامجك شيئًا من الـ UX..

كما يمكنك تصميم أدوات موجودة مسبقًا لكن مع تطويرات وتحسينات عليها، أو يمكنك ذلك لتخصيص الأدوات بشكل معين ترغب به.

آخر ما كتبته في مشواري مع هذا الكتاب – تقريبًا – هو هذا الباب، وذلك لآخذ بعين الاعتبار أفكارًا من منصات جاهزة ستجدها في الباب الرابع. وقد ترددت بالبداية هل أضع هذا الباب بعد الباب الذي يحوي المنصات الجاهزة أم قبله، فوضْعه قبله يعطي القارئ – إن قرأ الكتاب بتسلسله – مهلة لفهم تصميم الأدوات حتى لو لم يكن له خبرة معها؛ فإنه عندما ينتقل للتعامل مع الأدوات فسيكون خبيرًا بها، عارفًا بأصولها، فاهما الغاية منها، مقدّرًا الجهد المطلوب للحصول عليها. أما وضعه بعده – وهو ما أفضله – فسيعطي

القارئ لمحات كثيرة عن الأدوات والغاية منها وأهميتها على امتداد أنواعها، كما يهيئُه لفهم تصميم الأدوات أكثر وأكثر، فسيكون له اطلاع جيد في هذه الحالة.

ولكن، اخترت وضعه أولًا، بعد حيرة من أمري.. كما أني عوّلت على تنويهاتي أول الكتاب أن الكتاب يمكن ألا يقرأ بالتسلسل حيث يمكن للقارئ أن ينتقي الفصول أو الفقرات التي يرغب بقراءتها، فالكتاب ليس تفصيليًّا ولا أكاديميًّا.. هو خلاصة اطلاع وتجارب دامت أشهرًا..

وقبل أن نبدأ، أحب أن ألفت انتباهك إلى اصطلاح سيعني لنا الكثير خلال مشوارنا مع فصول هذا الباب، سنطلق على من يقوم بتأليف الأداة وتصميمها المؤلف Author، وعلى الشخص الذي يستخدم الأداة في مشاريعه المبرمج Programmer، وعلى الشخص الذي يستخدم الأداة ضمن منتج نهائي المستخدم User. وهذا ينطبق أيضًا على من يبرمج ملفات dll، فهناك من كتب الكود وهناك من استخدمه في مشاريعه وهناك من استخدمه كمنتج.

الفصل السادس – مدخل إلى تصميم الأدوات

قبل أن تدخل مجال تصميم الواجهات، أنصحك بفهم مبادئ تصميم الأدوات بشكل جيد، حتى لو لم تكن تنوي أن تنتج شيئًا منها.

أدوات المستخدم عن كثب

أدوات المستخدم أو الأدوات الخاصة هي أدوات معدلة عن أدوات Controls موجودة مسبقًا أو مضاف عليها. فعندما ننشئ أي أداة – كما سترى – فإننا نعتمد على أدوات سابقة، قد تكون مجرّدة وقد لا تكون، فنحن فعليًّا لا نأتي بأداة لا سلف لها، وإنما غالبًا نعتمد على أدوات سابقة.

وعند بناء أدوات كثيرة تشترك ببعض الأمور يفضل استخدام وإنشاء فئات مجردة. والفئات المجردة هي فئات يمكن الوراثة منها ولا يمكن استنساخها، والفائدة منها تكمن في تنظيم الكود وتسهيله عندما يكون مشتركًا مع فئات كثيرة. [1] خذ مثلًا مصطلح "العربة" أو "وسيطة النقل" كمثال واقعي، إذ يمكن اعتباره مصطلحًا مجرّدًا، بينما "السيارة" و"الطائرة" وغيرها من وسائط النقل تعتبر مصطلحات غير مجردة، لأنها أشياء حقيقية حسية يمكن تجربتها ومعاينتها. وكمثال برمجي لديك الفئة ButtonBase والتي تحوي أعضاءً [2] مشتركة لفئات مختلفة، فالأعضاء المشتركة بين الفئات Button وCheckBox وRadioButton تجدها في الفئة الأم ButtonBase، لذلك تم إنشاء الفئة الأخيرة ووضع الأعضاء المشتركة فيها وجعلها مجردة لكي لا يتم استنساخها، ثم تمت وراثتها من قبل الفئات الثلاثة المذكورة. ولولا ذلك لكان عليك إنشاء أعضاء كل فئة على حدة، وكتابة مئات وآلاف الأسطر المكررة المشتركة! [3]

حقيقة الأدوات Controls

الأدوات Controls هي فئات في الأساس، ويسري عليها ما يسري على الفئات، كجعلها غير قابلة للوراثة (كأدوات ProgressBar وImageList)[4]، فما سنقوم به هو تطوير فئات تمثل الأدوات الخاصة بنا.

يمكن أن تكون الأداة كائنًا فيزيائيًّا يأخذ حيّزًا معينًا (له حجم Size ولون Color و...) ويمكن أن تكون مجرد أسطر برمجية كـ Timer وOpenFileDialog وغيرها (الأدوات التي تضاف إلى أسفل بيئة التطوير عند التصميم) والتي تسمى مكونات Components.

بشكل عام، فإن أي كود يمكن أن يحفظ كأداة، وبعض الأدوات ما هي إلا مجموعة من الأكواد، فالأدوات غير الرسومية (المكونات) هي أكواد حفظت كأداة ما، ووضع لها خصائص عامة يمكن للمستخدم التفاعل معها للتأثير على ظروف عمل هذه الأداة. وبهذا، يمكنك

[1] انظر https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/abstract، وانظر كتاب "برمجة أطر عمل NET." لـ تركي العسيري (ص: 174).
[2] أعضاء الفئات في سي# هي كل ما يمثل الفئات من صفة (خاصية) أو فعل (طريقة)، انظر هذا الرابط https://www.techopedia.com/definition/25589/class-members-c-sharp
[3] مايكروسوفت – الفئة ButtonBase
https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.buttonbase?view=netframework-4.5
[4] انظر كتاب "برمجة أطر عمل NET." لـ تركي العسيري (ص: 572).

إنشاء هذه الأدوات (الأكواد) واستخدامها من خلال مصمم النموذج أو من خلال استنساخها ككائنات، فالمصمم أصلًا يقوم بذلك إذا أنشأتها من خلاله.

المكونات Components

المكونات Components مثلها مثل الأدوات Controls هي فئات Classes، إلا أن الاختلاف بينهما يكون بمصدر الوراثة، فالوراثة من فئة Control – أو ما يرث منها – يجعل فئتك أداة، بينما الوراثة من الفئة Component يجعلها مكونًا.

والفارق بين الأداة Control والمكون Component هو أن الأدوات ذات بنية رسومية، على عكس المكونات. وأنت – من حيث تدري أو لا تدري – تتعامل مع المكونات يوميًّا، فالمؤقتات Timers مثلا هي مكونات.

صعوبات العمل مع أدوات المستخدم

ستعاني في البدايات من صعوبة تمثيل أفكارك على شكل فئات لأدواتك الخاصة، ثم من صعوبة إيجاد أو تأليف الأكواد التي تنفذ فكرتك، ثم من صعوبة صيانة الكود وتنظيفه، فضلًا عن تطويره.

يمكنك التغلب على الصعوبة الأولى بالاطلاع على فئات الأدوات الخاصة المنتشرة هنا وهناك، عسى أن يكون هذا الكتاب أحد مصادرك. كما يمكن التغلب على الصعوبة الثانية بالتمرن والألفة والاعتياد على الفئات والبرمجة الكائنية OOP عمومًا. أما الصعوبة الثالثة فبالتجربة والتمرين على أدواتك نفسها.

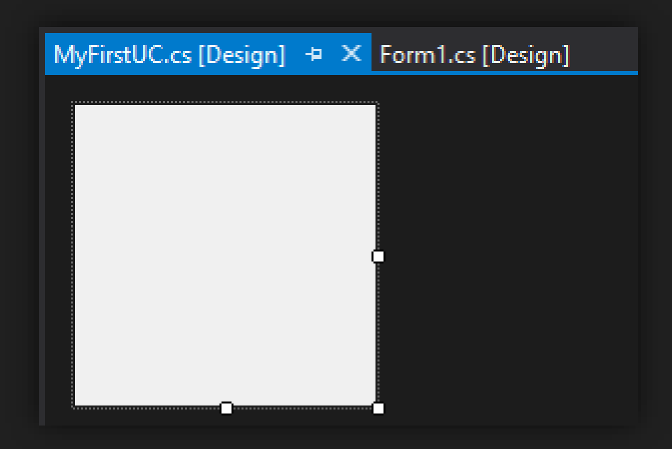
أساليب إنشاء أدوات المستخدم

غالبًا ما ننشئ الأدوات اعتمادًا على أدوات سابقة، حيث نقوم بوراثة أداة ما والتعديل عليها. كما يمكن دمج أكثر من أداة ضمن أداة جديدة واحدة، بحيث تصبح الأداة في هذه الحالة أشبه بنافذة تحضن الأدوات المدموجة مع بعضها (وهي ما تسمى بأداة المستخدم User Control). وأصعب أسلوب هو رسم الأداة من الصفر دون الاعتماد على أي أداة أو مكتبة أو منصة!

الأسلوب الأول شائع الاستخدام وهو أسهل الأساليب وأبسطها، وفيه يتم تطوير أداة ما لتحسينها أو التعديل عليها (التعديل على القيم الافتراضية لخصائصها مثلًا) أو حذف بعض الخصائص منها. أما الأسلوب الثاني فيُستخدم عند استخدام مجموعة من الأدوات معًا بصورة متكررة، وهنا لا يتم التعديل على الأدوات أو تطويرها بشكل مباشر. والأسلوب الأخير يُلزمك بإنشاء كل شاردة وواردة في الأداة، من أحداث وطرق وخصائص!

إنشاء أداة بإضافة ملف فئة Class للمشروع

يمكن إنشاء أداة مستخدم UserControl ما بإضافة ملف من النوع cs للمشروع والذي سيمثل فئة الأداة المراد إنشاءها. في أي مشروع WindowsForms عادي اختر Project ثم Add User Control ثم سمها MyFirstUC:



افتح كود هذه الفئة لتجده بالشكل التالي:



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace UserControlsTest
{
    public partial class MyFirstUC : UserControl
    {
```

```
        public MyFirstUC()
        {
            InitializeComponent();
        }
    }
}
```

لاحظ أن أداتنا الأولى وارثة من فئة UserControl، والتي تعطي الأدوات التي ترث منها مجموعة من الطرق والخصائص والأحداث القياسية. تأمل نافذة الخصائص والأحداث وقارنها بالأدوات القياسية الأكثر شعبية، لاحظ أن أغلب الخصائص والأحداث التي تتمتع بها عامة الأدوات موجودة هنا، مع غياب مجموعة أخرى من الخصائص والأحداث.

يمكنك جعل أداتك ترث من أدوات معينة، وذلك بالوراثة من فئاتها، ولتقوم بذلك قم بإضافة فئة Class للمشروع وسمها MySecondUC:

```
MySecondUC.cs    MyFirstUC.cs*    MyFirstUC.cs [Design]*    Form1.cs [Design]
UserControlsTest.MySecondUC
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Threading.Tasks;
 6
 7  namespace UserControlsTest
 8  {
 9      class MySecondUC
10      {
11      }
12  }
```

اجعل الفئة التي أضفتها ترث من الفئة Button على سبيل المثال:

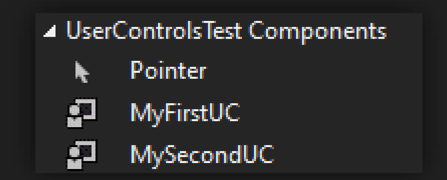
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace UserControlsTest
{
    class MySecondUC : Button { } }
```



عند الوراثة من أداة ما عليك التنويه بأنك تود استخدام مجال الأسماء الحاوي على هذه الأداة باستخدام الكلمة المحجوزة using.

أعد بناء المشروع من خلال الأمر Build Solution، وانتقل للنافذة Form1، ثم صندوق الأدوات ToolBox، لتحصل على الأداتين السابقتين:





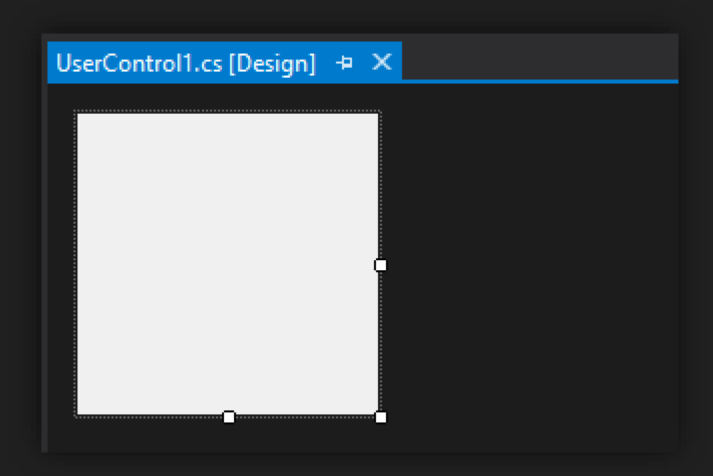
يمكنك بناء المشروع بالضغط على Ctrl+Shift+B.

على اعتبار أن MySecondUC ورثت من الفئة Button فإنها ستتضمن خصائصها وطرقها وأحداثها كاملةً، ومع هذا فلا فائدة من هذه الأداة، إذ إنها نسخة طبق الأصل عن الأداة الأولى، لا مميز فيها ولا شخصية لها، هي مجرد أداة Button عادية باسم مستعار.

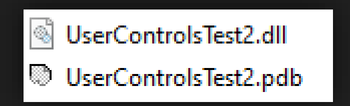
إنشاء أداة من خلال مشروع Windows Forms Control Library

عادة ما تجمع مجموعة من الأدوات ضمن مجال أسماء واحد، وهو مبدأ إنشاء المنصات Frameworks. هذه المنصات هي مكتبات ارتباط حيوي DLL، والتي تسند لمشاريع أخرى كمرجع Reference.

أنشئ مشروعًا من مشاريع مكتبات الأدوات Windows Forms Control Library – سمه UserControlsTest2 – لتحصل على أداة UserControl1 بشكل افتراضي (في مشاريع نماذج ويندوز ستحصل على نموذج):



كأي مشروع من النوع Library لا يمكنك تشغيل هذا المشروع من خلال الأمر Start أو عن طريق المفتاح F5، وعوضًا عن ذلك انقر على بناء المشروع، لتحصل على ملف dll يمثل الأداة – أو الأدوات – التي صممتها، ضمن المجلد bin ضمن المشروع:



في الواقع يمكنك إنشاء مشروع Class Library وإنشاء أدواتك فيه، فلا يوجد فرق جوهري بين هذا المشروع والمشروع السابق، اللهم إلا ما ستحصل عليه تلقائيًا عند إنشاء المشروع. فيمكنك القيام بإنشاء الأدوات أو الفئات العادية – التي لا تمثل أدوات Controls – من كلا المشروعين.

وسواء أستخدمت المشروع الأول أو الثاني فإنك ستحصل في نهاية المطاف على مكتبة dll ستسندها كمرجع لمشاريعك الأخرى.

تجربة الأدوات واختبارها، برمجة الأدوات

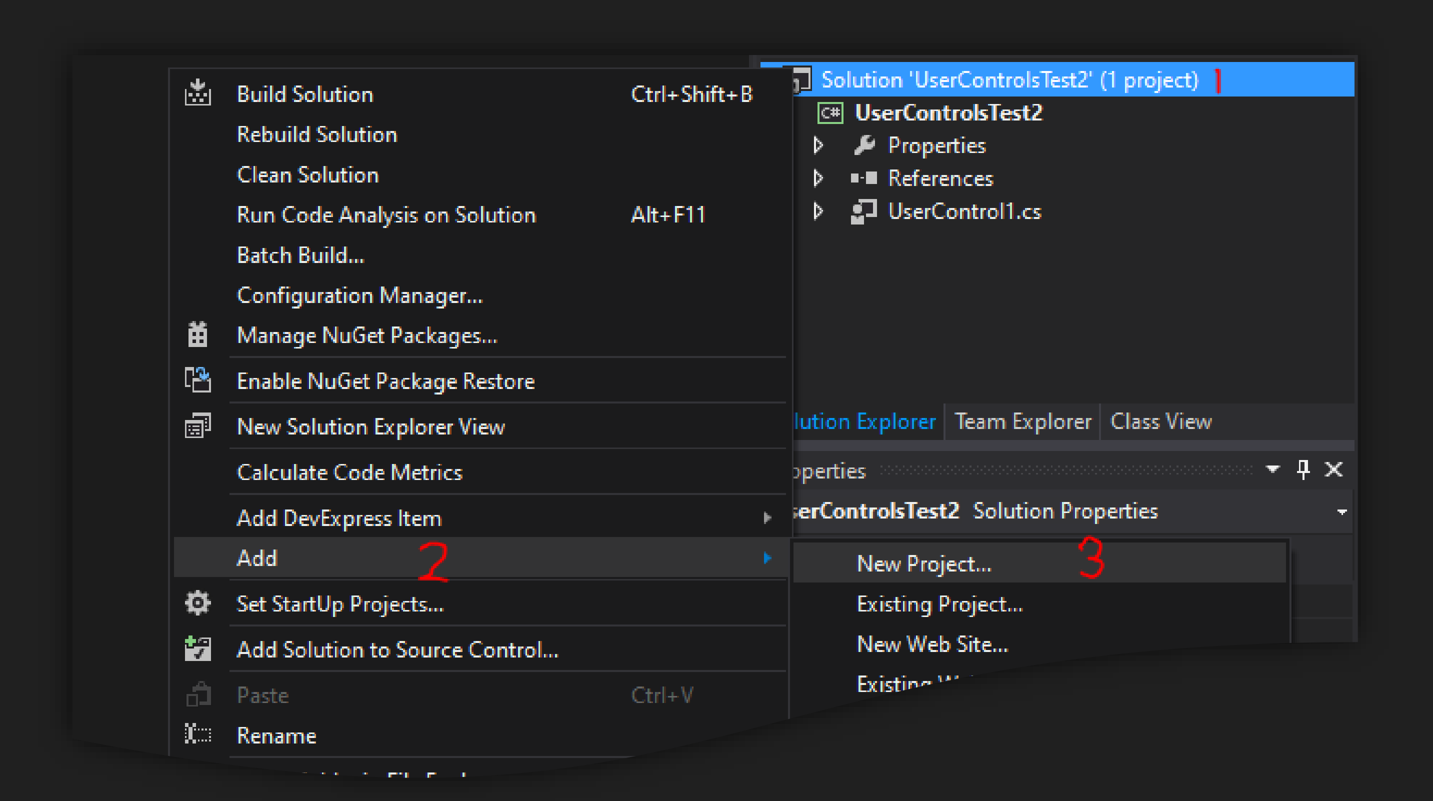
لا أقصد ببرمجة الأدوات إنشاءها – كما في الفقرتين السابقتين – وإنما برمجتها من قبل مبرمج (استخدامها ضمن مشاريعه)، هل تذكر ما اتفقنا عليه في مقدمة هذا الباب؟ من يقوم بإنشاء الأداة هو المؤلف، ومن يقوم باستخدامها ضمن مشاريعه هو المبرمج، ومن يقوم باستخدامها ضمن منتج نهائي هو المستخدم. وعليه فسنسمي عملية إنشاء

الأدوات تأليفها، وعملية استخدامها ضمن مشروع ما برمجتها.. وهذا هو الفرق الجوهري بين برمجة الأدوات بتأليفها وبرمجتها باستخدامها.

يمكنك اختبار وتجربة الأدوات – والتي ألّفتها كما اصطلحنا منذ قليل – من خلال إضافة مشروع WindowsForms جديد، ويمكن ذلك بإضافته بشكل مستقل عن مشروع الأدوات – سواءً كانت Windows Control Library أو Class Library – أو بشكل مدمج فيه.

لإضافة مشروع جديد مستقل انقر على File ثم New ثم Project ثم اختر مشروع نوافذ عادي، وهذا ما سيغلق المشروع السابق. ضمن مراجع المشروع References أضف مكتبة الأدوات الخاصة بك إذا كانت تحوي على فئات ستحتاجها مع أدواتك. وضمن صندوق الأدوات ToolBox أنشئ مجموعة جديدة ثم أضف لها مكتبة أدواتك الخاصة.

أما لإضافة مشروع مدمج مع مشروع مكتبة الأدوات Windows Control Library – أو الفئات Class – فاختر Add ثم New Project، وذلك بالنقر بالزر الأيمن على اسم المشروع ضمن مستكشف المشروع Solution Explorer (يمكنك أيضًا ذلك من خلال القائمة File ثم Add ثم New Project):



يفضل دمج المشاريع مع بعضها، لما يوفر عليك ذلك عناء التنقل بين المشاريع، كما أن المشاريع المدمجة ستتمكن من الحصول على أي تحديث للأدوات التي تم تطويرها وتحديثها مباشرة، أما لو تم استخدام المشاريع المنفصلة فقد تحتاج لنسخ مكتبة الأدوات في كل مرة تقوم بتحديثها، وهذا لن يعجبك على المدى البعيد!

خصائص وطرق وأحداث الأدوات

خصائص الأدوات (الفئات) Properties تعطي المبرمجين إمكانية ضبط الأدوات والتحكم بصفاتها، أما الطرق Methods فهي ما يمكن للأدوات أن تقوم به (قد تكون توابع وقد تكون إجراءات[1])، والأحداث Events هي الظروف التي تمر على الأدوات. هذه المكونات الثلاثة – وغيرها – هي أعضاء الفئات.

حتى يستطيع المبرمج الوصول لأعضاء أداة ما، يجب أن تكون عامة Public، قد ترغب – كمؤلف للأدوات – أن تمنع المبرمجين من الوصول لأعضاء معينة، وهذا ما سنراه في فقرة لاحقة.

سنعرف خاصيةً Property للأداة الخاصة MyFirstUC التي أنشأناها منذ فقرات، لذلك أضف هذه الأسطر البرمجية لفئة الأداة الخاصة هذه:



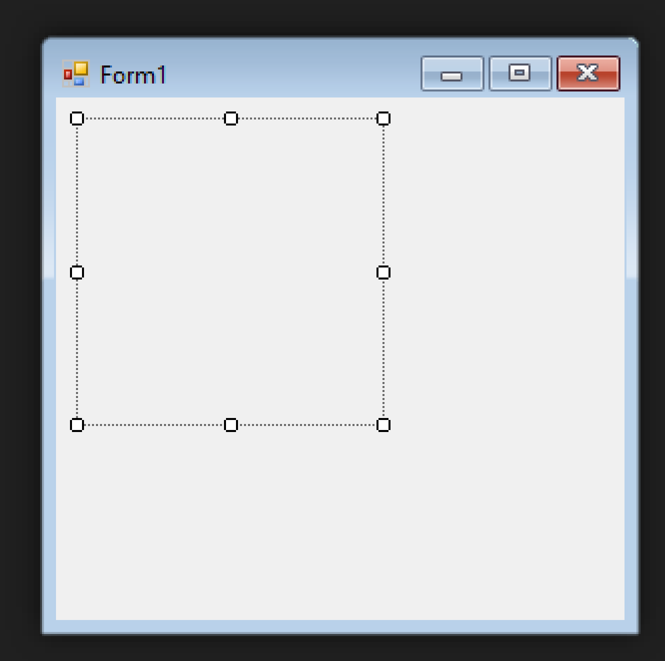
```
private string myproperty;

public MyFirstUC()
{
    InitializeComponent();
}

public string MyProperty
{
    get { return myproperty; }
    set { myproperty = value; }
}
```

[1] الإجراءات هي طرق لا تعيد قيمة، تعرّف بالكلمة المحجوزة void.

قم بإعادة بناء المشروع وانتقل للنافذة Form1، وأضف عليها أداةً من النوع MyFirstUC:



انتقل لنافذة الخصائص Properties window ولاحظ الخاصية التي أضفتها:

Modifiers	Private
MyProperty	
⊞ Padding	0, 0, 0, 0

يمكنك أيضًا الوصول للخاصية من خلال الأكواد:

```
public Form1()
{
    InitializeComponent();
    myFirstUC1.my
}
        MyProperty          string MyFirstUC.MyProperty
```

لاحظ أن MyProperty عبارة عن خاصية، وهذا واضح من الرمز ، أما لو جعلت المتغير myproperty عامًّا Public فإنك ستلاحظ وجود الرمز أمامه في لائحة أعضاء الكائن. عادة لا نجعل المتغيّرات عامة، وإنما نعتمد على الخصائص.

في كل مرة تقوم فيها بتعديل الأداة عليك إزالتها من النماذج التي تحويها أولًا، وبعدها تقوم بالتعديلات المطلوبة ثم إعادة بناء المشروع، ثم إضافتها للنماذج مجددًا. توقّع تعديل أدواتك عشرات المرات.

هذا بالنسبة للخصائص. أما الطرق، فهي توابع أو إجراءات عامة ولا تحتاج تفصيلًا. وأما الأحداث، فهي إجراءات تأخذ وسطاء محددة، سنفصّلها في فقرة لاحقة.

تفاصيل ستجعل أدواتك أنيقة – احترافية

على عكس ما سيُفهم من عنوان هذه الفقرة، فإني لا أقصد بالتفاصيل تلك التي ستعجب المستخدم، وإنما أقصد المبرمج الذي سيعتمد على أدواتك. صحيح أن بعض الفقرات الفرعية ضمن هذه الفقرة قد تلعب دورًا في جذب المستخدم، إلا أن المقصود الأول هو المبرمج.

وثّق ووصّف أدواتك

ناقشنا التوثيق بإيجاز في الفصل الثاني، وذكرنا أهميته وقيمته وما سيضيفه لفئاتك ومكتباتك، وفي هذه الفقرة سنناقش توصيف الأدوات (الفئات) والتحكم بمواصفاتها العامة.



المواصفات Attributes هي فئات Classes تأخذ في توابعها البنّاءة قيمة أو أكثر لوصف الأدوات.

يمكّنك مجال الأسماء System.ComponentModel من ضبط مواصفات أدواتك والتحكم بها، وكما وجدنا كثرة استخدام التوثيق ضمن فئات المحترفين (كميكروسوفت، مع حفظ الألقاب)، فإنك ستلاحظ في نهاية الفقرة أهمية التوصيف والغاية منه وشهرته.

كبداية، عدّل كود الأداة MyFirstUC ليصبح بالشكل التالي (احذف أي نسخة منها قبل التعديل إذا كنت قد أنشأتها في النافذة):

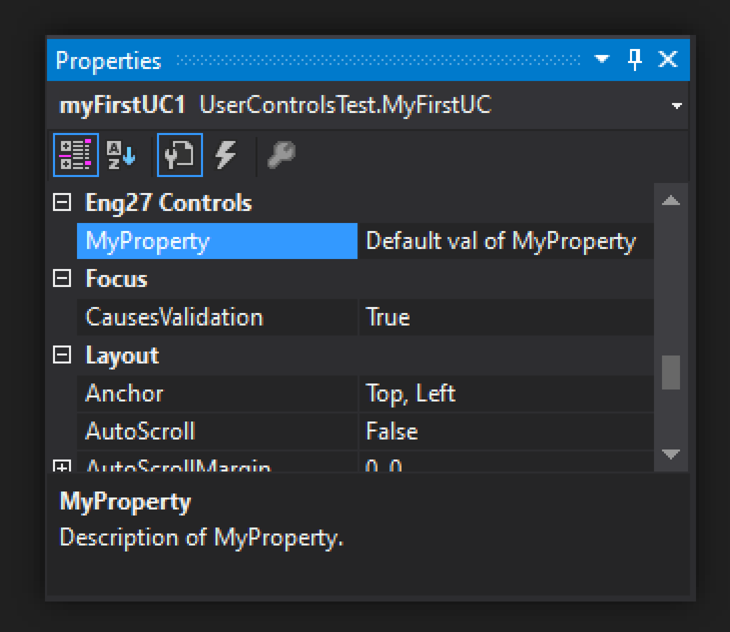


```
using System.ComponentModel;
using System.Windows.Forms;

namespace UserControlsTest
{
    public partial class MyFirstUC : UserControl
    {
        private string myproperty;
        public MyFirstUC()
        {
            InitializeComponent();
            myproperty = "Default val of MyProperty";
        }

        /// <summary>
        /// Gets or sets the MyProperty value.
        /// </summary>
        [Category("Eng27 Controls")]
        [DefaultValue("Default val of MyProperty")]
        [Description("Description of MyProperty.")]
        public string MyProperty
        {
            get { return myproperty; }
            set { myproperty = value; }
        }
    }
}
```

كما تلاحظ فإن توصيف أعضاء الفئات يكون بوضع الفئة المراد التوصيف على أساسها ضمن قوسين متوسطين [ ]، حيث يتم إرسال قيمة – قد تكون نصية وقد لا تكون – كوسيط لهذه الفئات. لاحظ أيضًا أننا أضفنا توثيقًا للخاصية MyProperty (من المفترض أن تعتبر ميكروسوفت أن المواصفة Description هي نفسها التوثيق Summary، حتى لا نكرر نفس الوصف لكل خاصية!). الآن، أعد بناء المشروع ثم أنشئ نسخة من الأداة في النافذة Form1، وانتقل لخصائصها، وغير نمط العرض لأقسام Categorized عوضًا عن العرض أبجديًّا Alphabetical وانتقل للخاصية المذكورة:



لاحظ أن القيمة الحالية للخاصية هي القيمة الافتراضية – على اعتبار أنها ليست بخط غامق – وأن الخاصية موجودة ضمن قسم Category معين، هذا فضلًا عن أنه بات للخاصية وصف يشرح وظيفتها. أما بالنسبة للتوثيق:

```
public Form1()
{
    InitializeComponent();
    myFirstUC1.my
}
                MyProperty        string MyFirstUC.MyProperty
                                  Gets or sets the MyProperty value.
```

باتت وظيفة وكيفية التوصيف واضحة أليس كذلك؟

إذا كانت إحدى خصائص أداتك الخاصة للقراءة فقط فلا داعي لإظهارها على نافذة الخصائص Properties window، فلن يتمكن المبرمج من تعديل قيمتها، لذلك أرسل القيمة false للتابع البنّاء Constructor للمواصفة Browsable كما يلي:

```
private readonly string myproperty2;

public MyFirstUC()
{
    InitializeComponent(); myproperty2 = "Default val of MyProperty2";
}
```

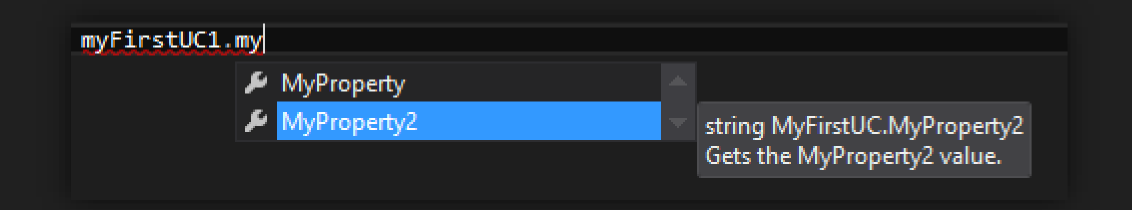
```
/// <summary>
/// Gets the MyProperty2 value.
/// </summary>
[Category("Eng27 Controls")]
[DefaultValue("Default val of MyProperty2")]
[Description("Description of MyProperty2.")]
[Browsable(false)]
public string MyProperty2
{
    get { return myproperty2; }
}
```

لا تنس حذف الأداة من النموذج قبل تعديلها.

قم بإعادة بناء المشروع ثم أنشئ نسخة من الأداة، لاحظ أن نافذة الخصائص لا تحوي الخاصية MyProperty2 لأنها متصفة بعدم قابلية الاستعراض Browsable(false):

⊟ Eng27 Controls	
MyProperty	Default val of MyProperty
⊟ Focus	
CausesValidation	True

لكن المبرمجين بالمقابل سيتمكنون من الوصول للخاصية من خلال محرر الكود، غير أنهم لن يتمكنوا من تغيير قيمتها، وهذا ما يخبرهم به توثيق الخاصية:



الموضوع ليس فقط إيعازًا بعدم إمكانية تغيير القيمة (وجود كلمة Get ضمن التوثيق دون الكلمة Set)، لو حاول المبرمج تغيير قيمة الخاصية فسيحصل على خطأ وسيدعو عليك:

2 Property or indexer 'UserControlsTest.MyFirstUC.MyProperty2' cannot be assigned to -- it is read only

لاحظ أن المواصفة Browsable لا تزيل الخاصية بالكامل من الأداة، هي فقط تخفيها عن طور التصميم (نافذة الخصائص Properties window) لكنها فعليًّا موجودة ويمكن الوصول إليها من طور التكويد (محرر الكود). سنتناول إزالة الخصائص والطرق والأحداث غير المرغوب بها في فقرة لاحقة.

أما إذا كانت الخاصية تتأثر بالإعدادات الإقليمية (كالخصائص النصية) فيفضّل إرسال القيمة true للمواصفة Localizable، مما يجعل الخاصية تختلف من لغة لأخرى. (إذا لم تتصف الخاصية بهذه المواصفة فإن قيمتها ستكون نفسها لكل اللغات).

ولضبط الخاصية الافتراضية والحدث الافتراضي للأداة، ضع المواصفتين DefaultProperty وDefaultEvent قبل فئة الأداة:

```
[DefaultProperty("MyProperty")]
[DefaultEvent("DoubleClick")]
public partial class MyFirstUC : UserControl
{
    //Class members
    //...
    //...
}
```

كما يمكنك ضبط أيقونة من النوع bmp بحجم 16 x 16 لتظهر على صندوق الأدوات ToolBox بجانب أداتك الخاصة عوضًا عن الأيقونة الافتراضية، وذلك من خلال المواصفة ToolboxBitmap. تم إعادة تعريف Overriding مشيد [1] المواصفة ليأخذ ثلاث حالات، الأولى مسار أيقونة الأداة الخاصة، والثانية نوع أداة موجودة مسبقًا لأخذ أيقونتها، والثالثة نوع أداتك الخاصة نفسها وتضمين الأيقونة ضمن المشروع.



ملفات bmp أفضل من ملفات ico لترميز الأدوات الخاصة، وذلك لأن الأيقونات ico لا تدعم إعادة التحجيم Resizing بشكل جيد.

[1] المشيد هو التابع البنّاء Constructor.

إذا بحثت عن هذه المواصفة، فستجد أن ميكروسوفت قد شرحت ذلك في مكتبتها MDSN بهذا الشكل[1]:

```
C#                                                        Copy

// Specifies the bitmap associated with the Button type.
[ToolboxBitmap(typeof(Button))]
class MyControl1 : UserControl
{
}
// Specifies a bitmap file.
[ToolboxBitmap(@"C:\Documents and Settings\Joe\MyPics\myImage.bmp")]
class MyControl2 : UserControl
{
}
// Specifies a type that indicates the assembly to search, and the name
// of an image resource to look for.
[ToolboxBitmap(typeof(MyControl), "MyControlBitmap")]
class MyControl : UserControl
{
}
```

ومع الأسف فإن الكود السابق – والشرح المرفق به – غير موفق ولا يليق بشركة كبيرة، فالكودان الأول والثاني غير عمليّان (الكود الثاني بالأخص)، لا بأس بضبط أيقونة نوع محدد من الكائنات كأيقونة لأداتك الخاصة، لكن هل من المنطقي أن تخبر المبرمج الذي سيستخدم مكتبة أدواتك أن يضع ملفات الأيقونات بالمسار الفلاني ثم يقوم بتضمين مكتبتك بمشروعه لتظهر أيقونات الأدوات؟! صحيح أن ميكروسوفت قصدت شرح كيفية استخدام المواصفة لا أكثر – ولا مشكلة عندي مع هذا – لكن الأهم والأكثر احترافية هو تضمين أيقونات الأدوات ضمن مشروعك نفسه الحاوي على الأدوات، وهذا ما لن تصل إليه باتباع شرح ميكروسوفت 🙂.

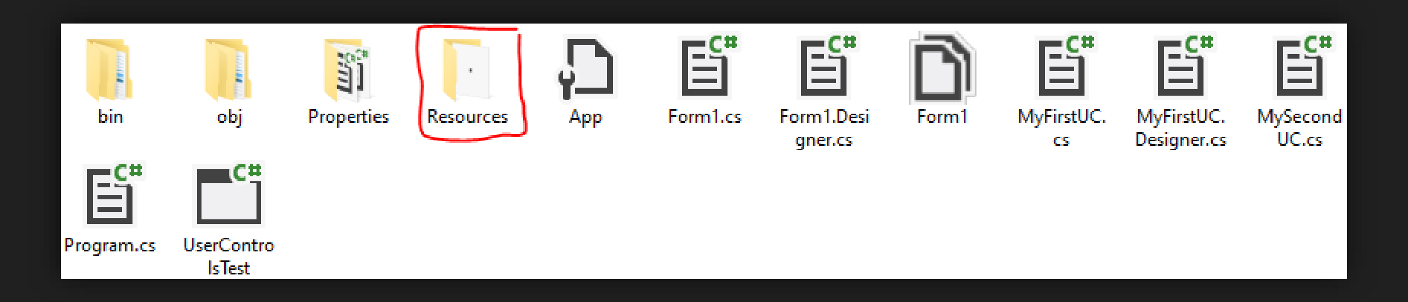
[1] انظر
https://docs.microsoft.com/en-us/dotnet/framework/winforms/controls/how-to-provide-a-toolbox-bitmap-for-a-control

الشيء الوحيد الذي أضمن أن ميكروسوفت تعرفه أن الأيقونة لن تظهر بالمشروع نفسه الذي أنشأت منه الأداة الخاصة، بمعنى أنه عليك إنشاء مشروع جديد وتضمين مكتبة أدواتك الخاصة لتحصل عليها – الأدوات الخاصة – مع أيقوناتها ضمن صندوق الأدوات ToolBox:

> ⓘ **Note**
>
> The bitmap does not appear in the Toolbox for autogenerated controls and components. To see the bitmap, reload the control by using the **Choose Toolbox Items** dialog box. For more information, see **Walkthrough: Automatically Populating the Toolbox with Custom Components**.

بعد البحث لساعات هنا وهناك، تبيّن أن هناك خطوات إضافية عليك القيام بها، لا أدري إذا كانت ميكروسوفت تعرفها أم لا، وهي كالتالي:

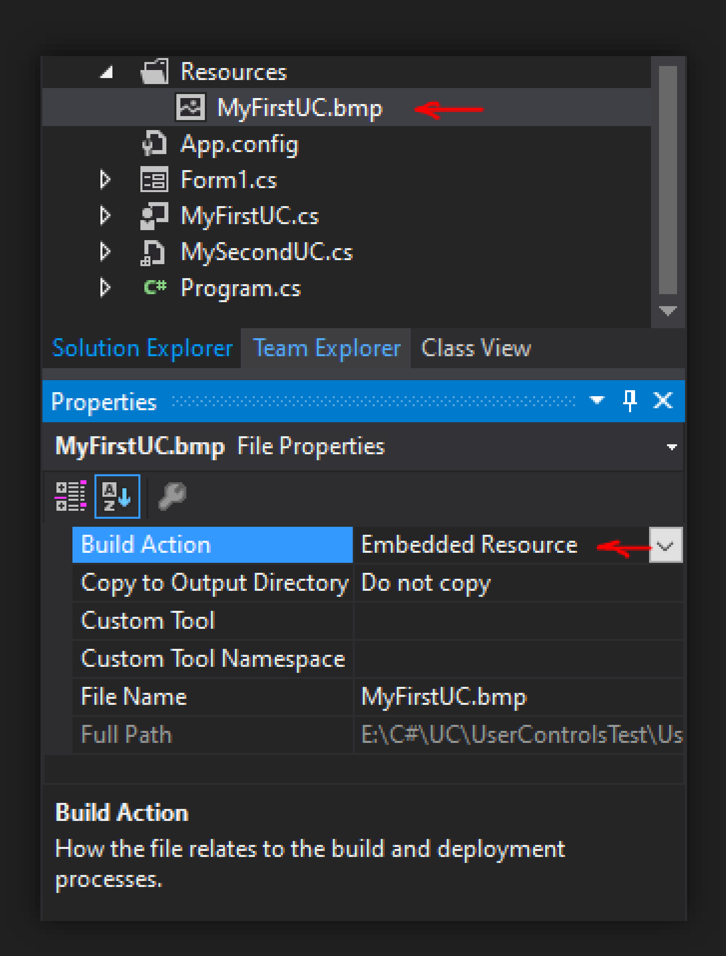
1. أنشئ أيقونةً من النوع bmp بحجم 16x16 وضعها في مصادر مشروعك (يمكنك تحويل أي صورة إلى النوع bmp من خلال برامج أو مواقع عديدة، مثل برنامج FormatFactory أو موقع convert-my-image [1]).
يجب أن تكون الأيقونة بنفس اسم أداتك كما يجب أن تكون بالمجلد الجذر لمشروعك وليس ضمن المجلدات الفرعية (وإلا يجب التنويه لذلك ضمن الكود)، في حالتي تم وضع الأيقونة في مجلد Resources:



[1] موقع تحويل الصور http://convert-my-image.com/ImageConverter.

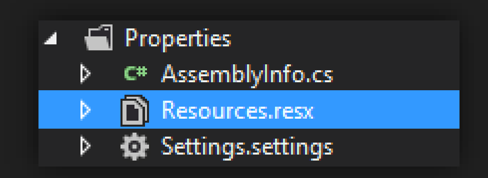


2. من صندوق الخصائص، اضبط خصائص الأيقونة كالتالي:

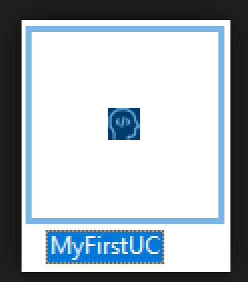


افتح مصادر المشروع أيضًا واختر الصورة التي أضفتها ثم غير الخاصية Persistance لـ Embedded in .resx:

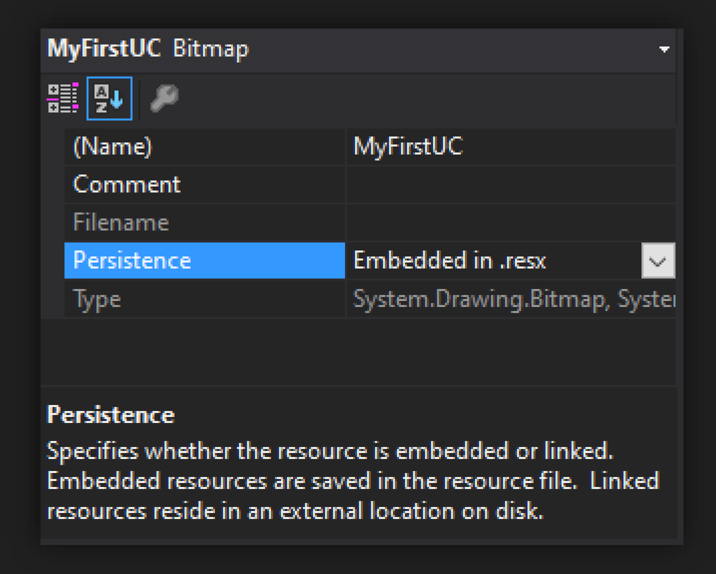
- افتح الملف Resources.resx:



- اختر صورة ما:



- غير خاصية Persistence لها:



3. استدع التابع البنّاء للمواصفة ToolboxBitmap بصيغته الثالثة، مرسلًا معه نوع فئة أداتك الخاصة ومسار الأيقونة (إذا كانت بالمجلد الجذر يكفي ذكر اسمها، وإلا، يجب كتابة مسارها كاملًا ضمن مجلدات مشروعك مستبدلًا الرمز \ بـ . "نقطة").
4. اكتب هذا التوصيف قبل بداية فئة أداتك الخاصة:

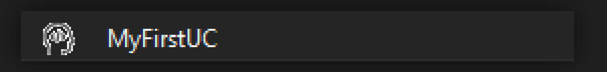
```
using System.Drawing;
//...

[ToolboxBitmap(typeof(MyFirstUC), "Resources.MyFirstUC.bmp")]
public partial class MyFirstUC : UserControl
{
    //Class members
    //...
    //...
}
```



المواصفة ToolboxBitmap موجودة ضمن مجال أسماء الرسومات System.Drawing، فليست كل المواصفات موجودة ضمن مجال الأسماء System.ComponentModel. كما أن بعض المواصفات موجودة ضمن مجال الأسماء System.

أعد بناء المشروع وأنشئ مشروعًا جديدًا من نوع نوافذ ويندوز WindowsForms وأضف مكتبة أدواتك [1] الحاوية على أداتك الخاصة التي ضبطت أيقونتها لصندوق أدوات المشروع الجديد، لاحظ تغير رمز أيقونة أداتك الخاصة من إلى لتصبح أداتك بالشكل التالي بعد تحميلها لصندوق الأدوات Toolbox:





إذا لم يتم عرض الأيقونة لخطأ ما، حاول تعديل فئة الأداة ضمن مشروعك وبالتحديد السطر الذي يحدد الأيقونة، وتأكد من وجودها (الأيقونة) ضمن ملفات المشروع بالمسار الذي حددته ضمن المواصفة ToolboxBitmap قبل إعادة بناء المشروع.

جرب أيضًا استخدام وضع البناء "تحرير" Release عوضًا عن تنقيح Debug، ثم ألغ تحميل الأداة من صندوق أدوات المشروع التي استخدمتها فيه وحملها مجدّدًا. (لاحقًا قم بالرجوع لوضع التنقيح Debug)

قد تواجه مشكلة في تحميل مكونات مكتبة الأدوات (بحجة أن المكتبة لا تحوي أدوات)، غيّر مكان المكتبة وحاول مجدّدًا.

[1] في حال كان مشروعك من النوع WindowsForms وأنشأت فيه الأدوات (كما في مشروعنا في هذه الفقرة) فإن الملف الناتج سيكون من نوع الملفات التنفيذية exe. يمكنك تغيير مخرجات مشروعك لـ dll من خصائص المشروع، وذلك بتغيير نوع الخرج Output type ضمن الصفحة Application لـ مكتبة فئات Class Library.

من المواصفات المفيدة – من بين عشرات المواصفات – المواصفة DebuggerDisplay الموجودة ضمن مجال الأسماء System.Diagnostics، والتي تظهر معلومات وتفاصيل معينة أثناء تنقيح البرنامج Debugging، ويستفاد منها عند وضع نقاط إيقاف BreakPoints على أسطر الكود أو عند حدوث أخطاء.

لفهم المواصفة أنشئ فئة مشابهة لهذه:



```
[DebuggerDisplay("This class represents person.")]
class Person
{
    public string FirstName;
    public string MiddleName;
    public string LastName;
}
```

استنسخها واضبط متغيراتها (على اعتبار أن المتغيرات عامة يمكن الوصول إليها من خارج الفئة)، ثم أنشئ نقطة توقف BreakPoint أو افتعل خطأً برمجيًّا كقسمة متغير عددي على آخر قيمته صفر، وانقل مؤشر الفأرة على كائن من هذه الفئة:

```
Person p = new Person();
p.FirstName = "Hasan";
p.MiddleName = "M.";
p.LastName = "al-Fahl";
   p  This class represents person.
```

في حال لم توصّف الفئة بهذه العبارة النصية فستحصل على:

```
Person p = new Person();
p.FirstName = "Hasan";
p.MiddleName = "M.";
p.LastName = "al-Fahl";
   p  {UserControlsTest.Person}
```

يمكنك أيضًا عرض بعض المعلومات عن الكائن:

p FirstName: "Hasan", MiddleName: "M.", LastName: "al-Fahl"

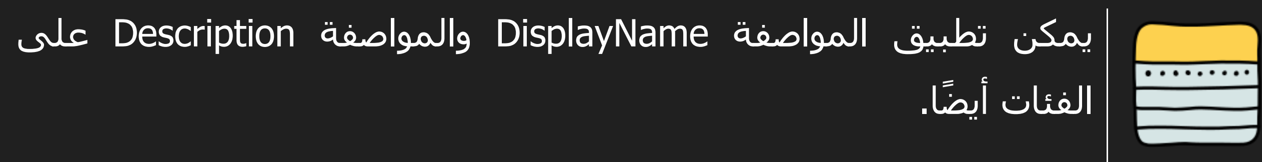
وذلك من خلال ضبط المواصفة DebuggerDisplay كالتالي (يمكنك كتابة الكود بسطر واحد):

```
[DebuggerDisplay(
    "FirstName: {FirstName}, MiddleName: {MiddleName}, LastName: {LastName}"
)]
```

وللمواصفة DisplayName الموجودة ضمن مجال الأسماء System.ComponentModel فائدة لطيفة وجميلة، فهي تعرض اسمًا مخصّصًا للخصائص غير اسم الخاصية، لاحظ:

Eng27 Controls
My 3rd Property
MyProperty

لاحظ وجود الفراغات في اسم الخاصية، حتى ميكروسوفت لا تعرف هذه الحركة 😂 (هلمرة مو مع حفظ الألقاب).

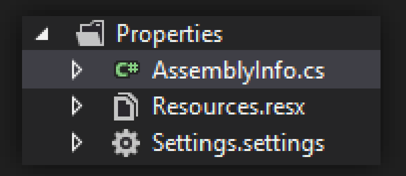



يمكن تطبيق المواصفة DisplayName والمواصفة Description على الفئات أيضًا.

كما أن المواصفة Flags الموجودة ضمن مجال الأسماء System تستخدم مع المعددات Enumerations، وسأحيلك لـ[مكتبة MSDN][1] إذ إن المثال المشروح فيها واضح ومفهوم.

[1] المواصفة Flag https://docs.microsoft.com/en-us/dotnet/api/system.flagsattribute?view=netcore-3.1.

أما المواصفة TypeConverter فإني أعتبرها أكثر المواصفات قيمةً فيما يتعلق بتصميم الأدوات، لذلك فسأخصص لها فقرة فرعيَّة لوحدها.



وأخيرًا، أختم مع مواصفات المجمّع Assembly، والتي يمكنك الوصول إليها من مجلد خصائص مشروعك ثم الفئة AssemblyInfo، والتي تحوي:



```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("UserControlsTest")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("UserControlsTest")]
[assembly: AssemblyCopyright("Copyright ©  2020")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to
COM
[assembly: Guid("8b732ce0-303d-4a5a-a4bb-727a7527f6da")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision
// Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

اتّق هفوات المبرمجين بالمعدّدات

من مبادئ تجربة المستخدم UX تقييد وتحديد اختيارات المستخدم ضمن قوائم أو لوائح لتوجيهه وتسهيل استخدام البرنامج عليه، صحيح أن من سيستخدم منتجاتك التي نناقشها في فصول هذا الباب – مكتبات dll، وأدوات خاصة – هم مبرمجون مثلهم مثلك، إلّا أن الخطأ وارد في البرمجة والتصميم، ولعلّك أدرى مني في هذا.

إذا كانت إحدى خصائص أدواتك تأخذ قيمًا بعينها، ولتكن "القيمة1" و"القيمة2" و"القيمة3"، فوجود قائمة منسدلة تحوي هذه القيم الثلاث أفضل من استخدام المتغيرات النصية وترك المبرمج وحده يفك ألغاز مكتبتك ويحاول اكتشاف القيم المناسبة لهذه الخاصية بعد رجوعه للشروحات Documentation التي ستزوده بها عن مكتبتك.

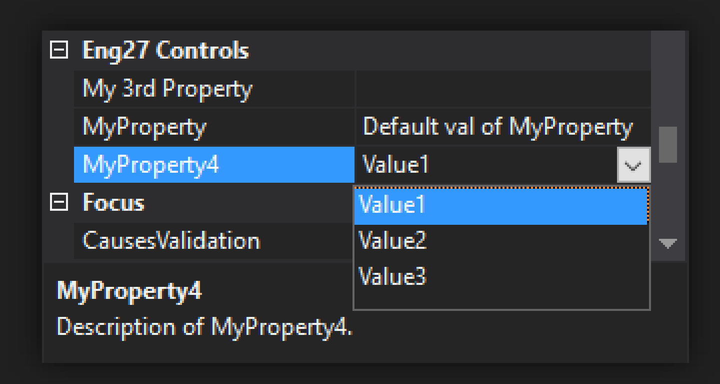
أضف هذه الأكواد للفئة MyFirstUC:



```
/// <summary>
///Specifies MyProperty4 values.
/// </summary>
public enum Values
{
    Value1,
    Value2,
    Value3
}

Values myproperty4;
/// <summary>
/// Gets or Sets MyProperty4 value.
/// </summary>
[Category("Eng27 Controls")]
[DefaultValue(0)]
[Description("Gets or Sets MyProperty4 value.")]
public Values MyProperty4
{
    get { return myproperty4; }
    set { myproperty4 = value; }
}
```

أعد بناء المشروع (بعد حذف أي نسخة من الأداة) ثم أنشئ الأداة مجدّدًا:



لاحظ أن القيمة الافتراضية للخاصية الجديدة هي Value1، وهي أولى قيم المعدد Values، وبالتالي فإن قيمته العددية 0 (والقيمة الثانية 1، والثالثة 2، ...)، كما أن الخاصية MyProperty4 لا تستقبل إلا القيم الثلاثة التي تقدمها القائمة المنسدلة.



من الضروري جعل محددات الوصول لأنواع المتغيرات المستخدمة في الفئات أعلى من محددات وصول الكائنات المستنسخة منها (من أنواع المتغيرات) أو مساوية لها. أي أن محدد وصول نوع البيانات Values يجب أن يكون أعلى أو يساوي محدد الوصول الكائن MyProperty4 المستنسخ من نوع البيانات Values. لذلك فنوع البيانات Values هو عام Public حصرًا.

اجمع الخصائص المتشابهة لأدواتك الخاصة مع بعضها

نوّهنا منذ فقرتين للخاصية TypeConverter وذكرنا أن لها أهميةً كبيرةً بالنسبة للخصائص.

بالاعتماد على المعدّدات Enumerations يمكنك تخصيص مجموعة محددة من القيم لإحدى خصائص أدواتك الخاصة، وبالاعتماد على المواصفة TypeConverter يمكنك تجميع عدة خصائص ضمن خاصية معينة، تمامًا كالخاصية Font وعائلتها، أو الخاصية Size وخواصها الفرعية Width وHeight. يدعى هذا الأسلوب بعرض الخصائص على شكل شجري.

حتى تستخدم المواصفة TypeConverter يجب أن تجعل الخاصية الأم فئةً، وتضع داخلها الخصائص الأبناء، وتضبط بعض المواصفات. سنستخدم الخصائص الأربعة التي أنشأناها سابقًا، وسنضيف خاصية جديدة هي الخاصية الأم.. من القائمة Project أنشئ فئة جديدة سمها Eng27PropertyType واكتب بداخلها الكود التالي:



```
using System;
using System.ComponentModel;

namespace UserControlsTest
{
    [RefreshProperties(RefreshProperties.Repaint)]
    [Browsable(true)]
    public class Eng27PropertyType
    {
        private string myproperty;
        private readonly string myproperty2;
        private string myproperty3;
        Values myproperty4;

        public Eng27PropertyType()
        {
            myproperty = "Default val of MyProperty";
            myproperty2 = "Default val of MyProperty2";
        }

        public override string ToString()
        {
            return "";
        }

        /// <summary>
        /// Gets or sets the MyProperty value.
        /// </summary>
        [DefaultValue("Default val of MyProperty")]
        [Description("Description of MyProperty.")]
        public string MyProperty
        {
            get { return myproperty; }
            set { myproperty = value; }
        }

        /// <summary>
        /// Gets the MyProperty2 value.
        /// </summary>
        [DefaultValue("Default val of MyProperty2")]
        [Description("Description of MyProperty2.")]
        [Browsable(false)]
        public string MyProperty2
        {
            get { return myproperty2; }
        }
```

```
        /// <summary>
        /// Gets or Sets MyProperty3 value.
        /// </summary>
        [DisplayName("My 3rd Property")]
        public string MyProperty3
        {
            get { return myproperty3; }
            set { myproperty3 = value; }
        }

        /// <summary>
        /// Specifies MyProperty4 values.
        /// </summary>
        public enum Values
        {
            Value1,
            Value2,
            Value3
        }

        /// <summary>
        /// Gets or Sets MyProperty4 value.
        /// </summary>
        [DefaultValue(0)]
        [Description("Description of MyProperty4.")]
        public Values MyProperty4
        {
            get
            {
                return myproperty4;
            }
            set
            {
                // التأكد من أن القيمة الممررة صالحة
                if (!TypeDescriptor.GetConverter(typeof(Values)).IsValid(value))
                {
                    throw new ArgumentException();
                }
                // إذا كانت القيمة الممررة صالحة يتم تنفيذ ما يلي
                myproperty4 = value;
            }
        }
    }
}
```

أما فئة MyFirstUC فاجعلها كما يلي:



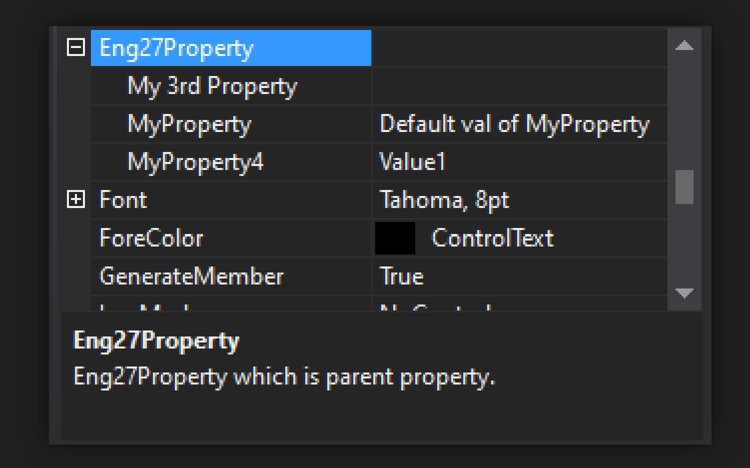
```
using System.ComponentModel;
using System.Windows.Forms;
using System.Drawing;
using System;
using System.Diagnostics;

namespace UserControlsTest
{
```

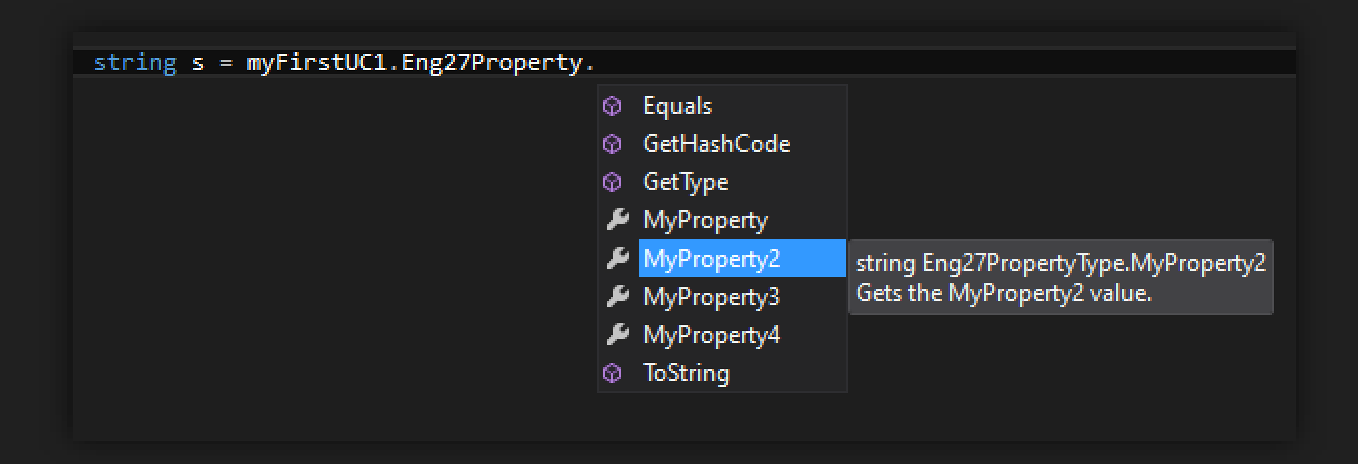
```
    [ToolboxItem(true)]
    [DefaultProperty("MyProperty")]
    [DefaultEvent("DoubleClick")]
    [ToolboxBitmap(typeof(MyFirstUC), "Resources.MyFirstUC.bmp")]
    public partial class MyFirstUC : UserControl
    {
        public MyFirstUC()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Eng27Property which is parent property.
        /// </summary>
        Eng27PropertyType eng27property = new Eng27PropertyType();
        [Browsable(true)]
        [TypeConverter(typeof(ExpandableObjectConverter))]
        [RefreshProperties(RefreshProperties.Repaint)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Content)]
        [Description("Eng27Property which is parent property.")]
        public Eng27PropertyType Eng27Property
        {
            get { return eng27property; }
            set { eng27property = value; }
        }
    }
}
```

أعد بناء المشروع وأضف الأداة MyFirstUC للنافذة، وانتقل لخصائصها:



لاحظ أن خصائص مشروعنا تجمعت تحت خاصية أم واحدة، وكما هو واضح فإنه لا يمكن للمبرمج الوصول إلى قيمة MyProperty2 من نافذة الخصائص Properties لأنها للقراءة فقط، لكن لا تقلق، لا زال بإمكانه الوصول إليها من خلال الكود:



لعلّك لاحظت إعادة تعريف الطريقة ToString وتساءلت عن السبب، هذه الطريقة – كما تعلم – تحوّل قيمة الكائن لقيمة نصية مهما كانت هذه القيمة، وبما أن الخاصية هي كائن – على اعتبارها فئةً – فعليك ضبط ما ستقوم به الطريقة ToString. إذا كنت قد أنشأت خاصيّة تحوي خصائص أبناء – بنفس أسلوب هذه الفقرة – وكانت الخصائص مجتمعةً – أو بعضها – تحمل معلومات تمثل الخاصية الأم؛ يمكنك إعادة قيمها بعد تنسيقها على شكل عبارة نصية ذات معنى بحيث تستفيد منها في مجريات برنامجك، فمثلًا إذا كانت لديك الخاصية الأم: FullName والتي تحوي الخصائص الأبناء: FirstName وMidName وLastName يمكنك إعادة تعريف الطريقة ToString لتعيد قيمة نصية بهذا الشكل:

```
public override string ToString()
{
    return FirstName + " " + MidName + " " + LastName;
}
```

كما يمكنك إنشاء خاصية عبارة عن معدّد تُظهر للمبرمج مجموعة من الخيارات التي تحدد محتويات القيمة التي ستعيدها الطريقة ToString، بحيث تظهر الاسم الأول والأخير مثلًا، أو الاسم الأول والأوسط، وغيرها.. للمزيد راجع هذا الرابط.[1]

[1] استخدام المواصفة TypeConverter
http://csharphelper.com/blog/2014/09/use-a-type-converter-with-a-propertygrid-control-in-c/

تخلّص من أعضاء الفئات غير المطلوبة

لا تحتاج دائمًا كل الخصائص – والأحداث والطرق – التي ترثها من فئات سابقة، بل قد تضرّك أحيانًا! لذلك عليك عدم تضمينها في فئاتك. لا شكّ أن بالونًا أصفر اللون قد تشكل بجانب رأسك وبداخله العبارة "Browsable"، لكن مع الأسف فهذه المواصفة لا تزيل أعضاء الفئات منها وإنما تخفي الخصائص والأحداث عن نافذة الخصائص والأحداث فقط (أي عن المصمّم)، ولا زال أمام المبرمج إمكانية الوصول إلى تلك الأعضاء التي أخفيتها من خلال محرر الكود.

في الواقع فإن الخاصية تظهر في القائمة Intellisense (القائمة المنبثقة التي تظهر وفيها أعضاء الكائنات بعد كتابة نقطة أمام اسم الكائن، والتي تكمل عنك كتابة أسماء أعضاء الكائنات إذا أدخلت أحرفها الأولى)، ويمكنك إخبار التقنية Intellisense هذه بعدم إظهار الخاصية – أو الطريقة أو الحدث – الفلانية ضمن القائمة المنبثقة في محرر الكود من خلال المواصفة EditorBrowsable؛ بتمرير القيمة EditorBrowsableState.Never للتابع البناء لهذه المواصفة. فمثلًا، إذا كانت أداتك لا تدعم الخاصية AutoScroll فإنه من المعيب إظهارها للمصمم وللمحرر:



```
[Browsable(false)]
[EditorBrowsable(EditorBrowsableState.Never)]
public new bool AutoScroll { get; set; }
```

أعد بناء المشروع وأنشئ نسخة جديدة من الأداة، كما هو متوقّع فإن الأداة لن تظهر في نافذة الخصائص لكنها – خلاف كل التوقّعات – ما زالت تظهر في محرر الكود:

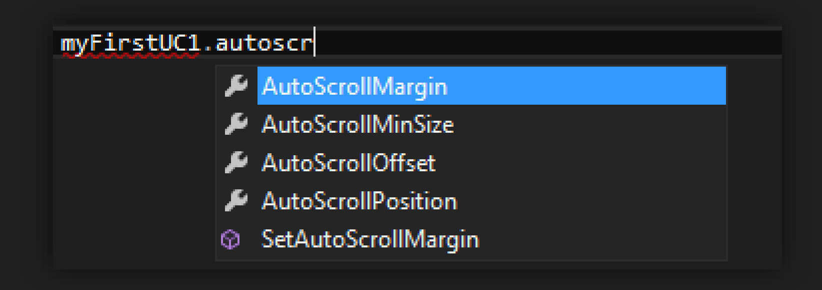
```
myFirstUC1.aut
    AutoScaleDimensions
    AutoScaleMode
    AutoScroll              bool MyFirstUC.AutoScroll
    AutoScrollMargin
    AutoScrollMinSize
    AutoScrollOffset
    AutoScrollPosition
    AutoSize
    AutoSizeChanged
```

مزعجة، أليست كذلك؟! ميكروسوفت قالت حول هذا:

> ⓘ **Note**
>
> In Visual C#, EditorBrowsableAttribute does not suppress members from a class in the same assembly.

وهذا يعني أن هذه المواصفة لن تخفي الخاصية عن الفئة إذا كانت في نفس المجمّع، وإنما ستخفيها عند إسنادك للفئة كمرجع – على شكل ملف dll – لمشروع آخر. (لا تنس إخفاء الخصائص المتعلقة بالخاصية المزالة، والتي هي في مثالنا AutoScroll).

هذا الكائن تم استنساخه في مشروع آخر:



قد تتساءل: لماذا لا نعيد تعريف الخصائص ونجعلها خاصة private، وأجيبك: لأنها ستظهر مجدّدًا، إذ إن للنسخ الأصلية العامة من الخاصية الأولوية!

يمكنك أيضًا إهمال الأعضاء بوصفها بالمواصفة Obsolete، لكنها لن تزيل الأعضاء من الفئات وإنما ستجعلها مهملة لا يمكن استخدامها (لو حاول المبرمج استخدام هذه الأعضاء سيعطي الفيجوال ستوديو رسالة خطأ للمبرمج، والمبهج في الموضوع أنه يمكنك تحديد هذه الرسالة 🌚).

إذا كانت لديك مجموعة كبيرة من الخصائص تودّ إخفاءها من المصمم يمكنك إنشاء فئة تقوم بذلك عوضًا عن توصيف كل الفئات المراد إخفاؤها بالمواصفة Browsable، ولذلك أنشئ الفئة MyFirstUCDesigner وأضف إليها المرجع System.Design (من خلال الأمر References > Add Reference):



```
using System.Collections;
using System.ComponentModel.Design;
using System.Windows.Forms.Design;

namespace UserControlsTest
{
    class MyFirstUCDesigner : ControlDesigner
    {
        protected override void PostFilterProperties(IDictionary Properties)
        {
            Properties.Remove("BackgroundImage");
            Properties.Remove("AutoScroll");
            Properties.Remove("AutoScrollMargin");
            Properties.Remove("AutoScrollMinSize");
        }
    }
}
```

ثم أضف المواصفة Designer لفئة أداتك الخاصة ممرّرًا لها فئة التصميم التي أنشأتها منذ أسطر:



```
[Designer(typeof(MyFirstUCDesigner))]
public partial class MyFirstUC : UserControl
{
    //Class members
    //...
    //...
}
```

كل ما سبق هو عبارة عن إخفاء الخصائص عن المصمم وعن المحرر، لكن في الواقع هذه الخصائص – وبقية أعضاء الفئة – موجودة ويمكنك كتابتها بشكل يدوي (عند نسخ الكود مثلًا). صحيح أن التقنية Intellisense ستحاول منعك، لكن المترجم ومحرر الكود لن يعترفا لا بك ولا بالتقنية وسيترجما ويحررا الكود:

```
myFirstUC1.AutoScroll = true;
```

لاحظ أن المحرر تقبّل الخاصية ولم يظهر رسالة خطأ، وإذا نفذت البرنامج (بالنقر على F5) سيعمل البرنامج دون أخطاء.

للمزيد، أحيلك إلى الروابط التالية:

- سؤال في موقع vbcity [1].
- سؤالان في موقع stackoverflow [2].
- مقال في موقع codeproject [3].

ميّز الخصائص المهمة

لكل أداة خصائص تميّزها، وهي أكثر الخصائص التي تعبّر عن الأداة ومحتواها والغاية منها. ولا أعني بهذا أن تجعل الخاصية المهمة ضمن الأداة خاصية افتراضية، فهذا شيء مفروغ منه، ولكنني أقصد إظهار الخصائص الرئيسية ضمن لوحة مهام Task Panes أو ما يسمى بقائمة الأوامر المميزة Smart Tag.



الترجمة الأفضل لـ Smart Tag هي العلامة الذكية، ولكنّي وجدت أن "الأوامر المميزة" يشير إلى وظيفة المصطلح الإنكليزي بشكل أفضل، خصوصًا أني لم أجده ضمن الكتب العربية.

[1] سؤال في موقع vbcity http://vbcity.com/forums/t/105290.aspx.
[2] سؤالان في موقع stackoverflow
https://stackoverflow.com/questions/1528371/hiding-unwanted-properties-in-custom-controls
https://stackoverflow.com/questions/32353708/remove-properties-and-events-from-usercontrol-vb-net
[3] مقال في موقع codeproject
https://www.codeproject.com/Articles/829337/Remove-Unwanted-Properties-and-Events-from-UserCon

عد إلى الفئة MyFirstUCDesginer، والتي كان مصادرها ثلاثة مجالات أسماء وأضف لها الأسطر البرمجية التالية:



```
[System.Security.Permissions.PermissionSet
 (System.Security.Permissions.SecurityAction.Demand, Name = "FullTrust")]
class MyFirstUCDesigner : ControlDesigner
{
    ...
    ...

    private DesignerActionListCollection actionLists;


    // ملء لائحة المهام بالأدوات
    public override DesignerActionListCollection ActionLists
    {
        get
        {
            if (null == actionLists)
            {
                actionLists = new DesignerActionListCollection();
                actionLists.Add(new MyFirstUCActionList(this.Component));
            }
            return actionLists;
        }
    }
}
```

كان بالإمكان التصريح عن مرجع للفئة من خلال الكلمة المحجوزة using لكن لحاجة في نفسي لم أفعل ذلك.

أنشئ فئة جديدة سمّها MyFirstUCActionList الموجودة داخل مجال الأسماء System.ComponentModel.Design:



```
using System;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Drawing;

namespace UserControlsTest
{
    class MyFirstUCActionList : DesignerActionList
    {
        private MyFirstUC myuc;
        private Eng27PropertyType mycustomtype;

        private DesignerActionUIService designerActionUISvc = null;
```

```
        // يقوم التابع البناء هذا بربط لائحة المهام بالأداة
        public MyFirstUCActionList(IComponent component)
            : base(component)
        {
            myuc = component as MyFirstUC;
            mycustomtype = new Eng27PropertyType();

            // مهمة السطر التالي إعادة تحديث محتويات لائحة المهام كلما تغيرت الخصائص
            this.designerActionUISvc = GetService(typeof(DesignerActionUIService))
              as DesignerActionUIService;
        }

        // تابع يقوم بجلب الخاصية من خلال اسمها
        private PropertyDescriptor GetPropertyByName(String propName)
        {
            PropertyDescriptor prop;

            // لاحظ أنه عليك تزويد التابع بالفئة التي تحوي الخاصية
            // في حالتنا هناك فئتان تحويان الخصائص التي تلزمنا في لائحة مهامنا
            // و هي فئة الأداة وفئة نوع البيانات الخاص بنا
            if (propName == "MyProperty")
                prop = TypeDescriptor.GetProperties(mycustomtype)[propName];
            else if (propName == "MyProperty4")
                prop = TypeDescriptor.GetProperties(mycustomtype)[propName];
            else
                prop = TypeDescriptor.GetProperties(myuc)[propName];
            if (null == prop)
                throw new ArgumentException("Property not found!", propName);
            else
                return prop;
        }

        // هذه الخصائص التي سنظهرها في لائحة المهام
        // عليك كتابتها كلها هنا

        // مثال على خاصية يمكن الحصول عليها من فئة الأداة (على اعتبارها ترث من فئة الأدوات الخاصة التابعة لويندوز)
        public Color BackColor
        {
            get
            {
                return myuc.BackColor;
            }
            set
            {
                GetPropertyByName("BackColor").SetValue(myuc, value);
            }
        }
        // مثال على خاصيتان لا يمكن الحصول عليها من فئة الأداة، وإنما من فئة أخرى لذلك عليك كتابة ذلك بشكل صريح
        public string MyProperty
        {
            get
            {
                return myuc.Eng27Property.MyProperty;
            }
            set
            {
              GetPropertyByName("MyProperty").SetValue(myuc.Eng27Property,value);
            }
        }
```

```
        public Eng27PropertyType.Values MyProperty4
        {
            get
            {
                return myuc.Eng27Property.MyProperty4;
            }
            set
            {
              GetPropertyByName("MyProperty4").SetValue(myuc.Eng27Property,value);
            }
        }

        // هذا التابع يقوم بإضافة العناصر للائحة المهام
        public override DesignerActionItemCollection GetSortedActionItems()
        {
            DesignerActionItemCollection items =
                new DesignerActionItemCollection();

            // إضافة عنوان لمجموعة من العناصر
            items.Add(new DesignerActionHeaderItem("WinForms Properties"));

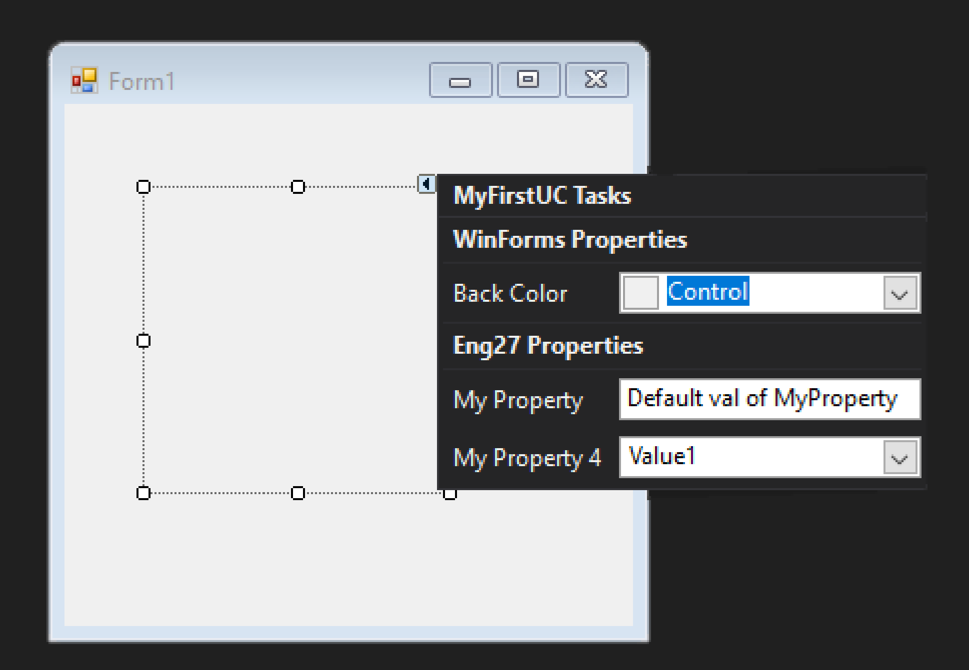
            // إضافة عناصر
            items.Add(new DesignerActionPropertyItem("BackColor",
                                 "Back Color", "Appearance",
                                 "Selects the background color."));

            // إضافة عنوان لمجموعة من العناصر
            items.Add(new DesignerActionHeaderItem("Eng27 Properties"));

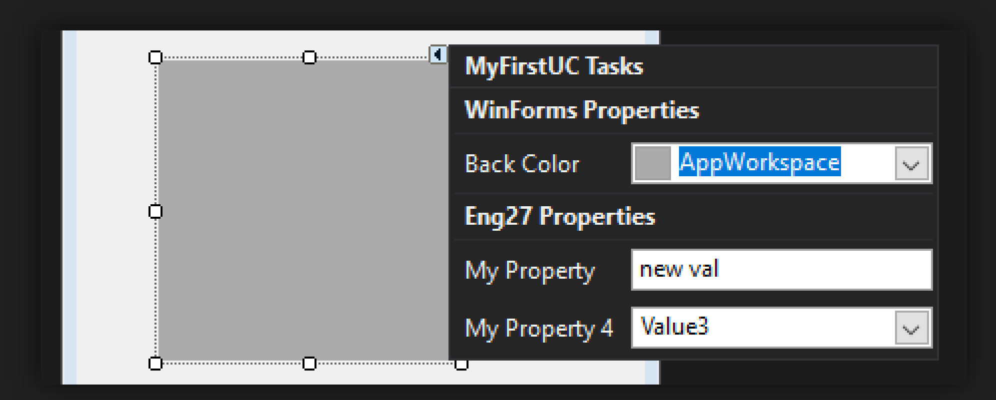
            // إضافة عناصر
            items.Add(new DesignerActionPropertyItem("MyProperty",
                                 "My Property", "Misc",
                                 "Selects the MyProperty text."));
            items.Add(new DesignerActionPropertyItem("MyProperty4",
                                 "My Property 4", "Misc",
                                 "Selects the MyProperty4 enum."));
            return items;
        }
    }
}
```

لاحظ أن العناصر تضاف بتمرير اسمها، والاسم الذي ترغب بأن تعرض عليه، والقسم Category المحتواة ضمنه، وقيمة نصية تصفها.

أعد بناء المشروع وأنشئ نسخة جديدة من الأداة، لاحظ وجود سهم صغير أعلى يمين الأداة ◨، انقر عليه:



غير الخصائص:



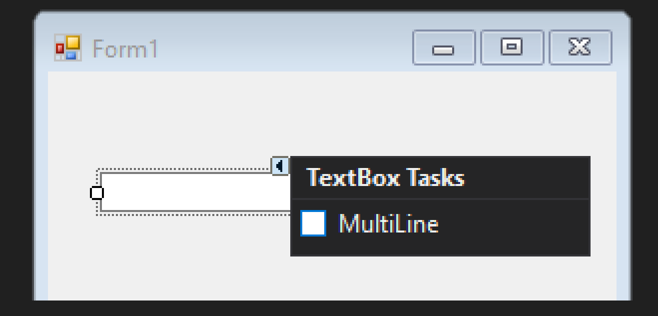
لاحظ نافذة الخصائص Properties window:

BackColor	AppWorkspace

Eng27Property	
My 3rd Property	
MyProperty	new val
MyProperty4	Value3

لاحظ أن الخطوات البسيطة السابقة أعطتك إمكانيات احترافية، قوية، لطيفة، متقنة.. فكل تفصيل صغير تضفيه على مكتبتك يغنيها ويثريها ويزيد قيمتها، لكن مع ذلك يجب أن تتريث ولا تكثر من التفاصيل إلا التي أحطتها من كافة الجوانب، فما قيمة مكتبة أدوات برمجية طويلة عريضة مليئة بالتفاصيل والأخطاء أيضًا؟ أداة بسيطة خفيفة مدروسة ومتكاملة خير من أداة فخمة مزيّنة تُخرج المبرمجين من الملّة بأخطاءها!

قد لا تحتاج لائحة المهام مع بعض الأدوات المشتقة من أدوات لها لوائح مهام. فمثلًا الأداة TextBox تعطيك لائحة فيها الخاصية MultiLine لجعلها صندوق نص متعدد الأسطر، بينما قد لا تحتاجها أحيانًا (كصناديق نصوص كلمات السر أو أي محتوى سطري):



إذ إنك حتى لو أخفيت الخاصية MultiLine من خلال المواصفة Browsable فإنها ستبقى ظاهرة في لائحة المهام، فإخفاء الخصائص لا يزيلها كما تقدّم.

في مثل هذه الحالة عليك إلغاء لائحة المهام لهذه الأداة، وذلك من خلال إنشاء فئة أداة جديدة مشتقة من هذه الأداة وإضافة المواصفة Designer لها:



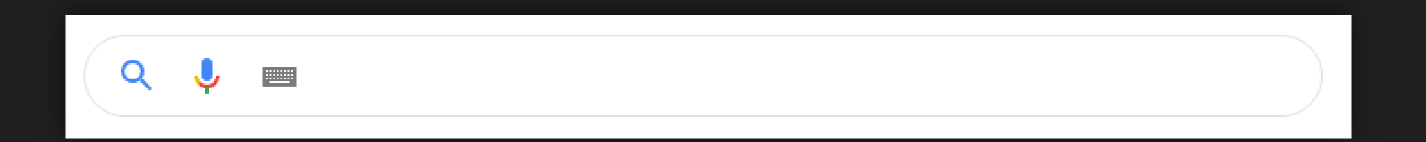
```
[Designer(@"System.Windows.Forms.Design.ControlDesigner, System.Design," +
          " Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a")]
```

للمزيد، أحيلك إلى مقال [1] وتلميح [2] من موقع codeproject عن لوائح المهام.

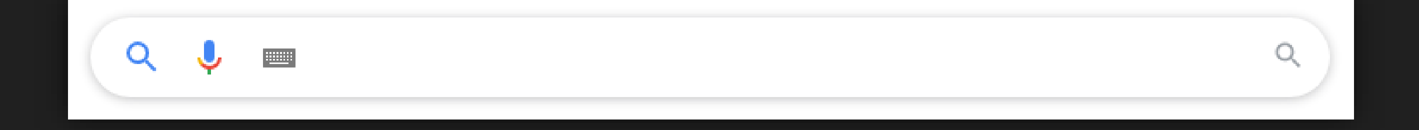
ظلّل أدواتك

يضفي الظل على الأدوات تأثيرًا جميلًا، يبثّ فيها الحياة ويغنيها بالتفاصيل، بدءًا من إعطاءها أشكالًا ثلاثية الأبعاد، مرورًا بتجميل الأدوات، وحتى تمييز الأدوات وتحديدها.

وقد يخطر ببالك أن الأدوات ثلاثية الأبعاد باتت موضة قديمة والاهتمام الآن بالأدوات ثنائية الأبعاد، وهذا صحيح، ولكنه ليس ما عنيته، فحتى الأدوات ثنائية الأبعاد يمكنك إعطاءها تأثيرًا ثلاثي الأبعاد – من خلال الظل مثلًا – وجعلها تبدو أعلى من السطح الحاضن لها (لا أقصد طبعًا جعلها تبدو بشكل ثلاثي الأبعاد). لاحظ مثلًا صندوق بحث غوغل كيف يكون قبل وصول التركيز إليه:



وكيف يكون بعد وصول التركيز إليه:



[1] مقال – تخصيص أدوات المستخدم بميزة لائحة المهام
https://www.codeproject.com/Articles/37103/Customizing-User-Controls-with-Smart-Tag-Feature
[2] تلميح – إخفاء بعض الخصائص والمهام للفئات المشتقة
https://www.codeproject.com/Tips/56028/Hiding-Inherited-Properties-and-Tasks-in-Derived-C

سنناقش الظل والتظليل لاحقًا ضمن فقرات هذا الفصل بأمثلة واقعية.

زوّد الأدوات بخصائص إضافية

هل تساءلت يومًا كيف يعرف الفيجول ستوديو أن أداة ToolTip تم إضافتها ليضيف خاصية ToolTip On toolTip1 إلى أدوات المشروع؟

أردت فقط في هذه الفقرة إثارة فضولك عن هذه الميزة ولن أناقشها هنا، وسأتركها لفقرة لاحقة، علّك تجد طريقةً للوصول إليها قبل أن تصل للفقرة التي ستتحدث عن الموضوع.

لكن قبل أن تنتقل إلى الفقرات التالية، فإني أود أن ألفت انتباهك إلى أن الخصائص التي تضاف للأدوات (خاصية ToolTip On toolTip1) ليست في الواقع تابعة للأدوات التي تمت إضافتها إليها، بمعنى أنه لو تم إضافة خاصية ما للأداة أ بفعل الأداة بـ، فإن هذه الخاصية لا علاقة لها لا من قريب ولا من بعيد بالأداة أ، ولا يمكن الوصول لها من خلال الأداة أ، وإنما هي أحد أعضاء الأداة بـ. (مزيد من التوضيع لاحقًا).

اعتمد على الفئة ControlPainter لرسم أدواتك من الصفر

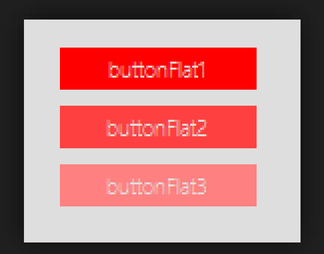
إذا كانت لديك الجرأة الكافية لإنشاء أداة من أدواتك من الصفر، أو حتى مجرد رسمها، فالفئة ControlPainter هي ما يجب عليك الاعتماد عليه. ويدخل ضمن ذلك النوافذ Forms نفسها، فالفئة هذه ترسم أزرار التحكم الخاصة بالنوافذ (التكبير، والتصغير، والإغلاق)، وغيرها.

ضع في ذهنك أنه عليك تزويد الأداة بالكيفية التي ستُرسم بها من خلال هذه الأداة، وإلا فستكون جامدة.

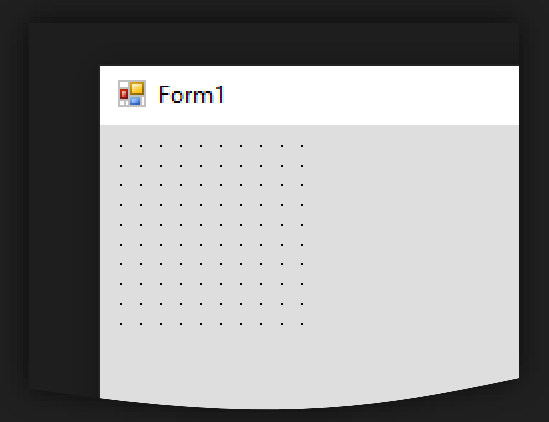
للفئة ControlPainter إجراءات كثيرة لرسم الأدوات القياسية، مثل الأزرار وصناديق الاختيار وغيرها، وهي لا تحتاج إلا لكائن رسومات Graphics ومستطيل يمثل حجم الأداة وموقعها.

جميع أعضاء الفئة ستاتيكية static، أي أنه يمكنك الوصول لهذه الأعضاء دون استنساخ الفئة.

وعلى سبيل المثال، التابعان Light وLightLight يعطيانك ألوانًا خفيفةً من ألوانٍ تمررها كلونٍ أساس، لاحظ عندما نجعل لون الخلفية للزر الأول أحمر، ولون الخلفية للزر الثاني والزر الثالث ما يعيده التابعان Light وLightLight:



كما أن الإجراء DrawGrid يرسم لك شبكة بالخلفية مكونة من مجموعة من النقاط:



عليك تمرير كائن رسومات، ومستطيل يمثل منطقة الرسم، وحجم الفراغات بين النقاط، ولونها، كوسطاء للإجراء DrawGrid.

لا تتكلف!

بعد كل التفاصيل التي تناولناها في الفقرات السابقة، لا تحاول تطبيق أي منها ما لم تتمكن منها. لا تتحفنا ببرامج فيها ما هب ودب من التفاصيل المتضاربة وغير المرتبة والمليئة بالأخطاء!

لا تفرح على برنامجك عندما تملؤه بتفاصيل تأخذها من هنا أو هناك دون أن تفهم أفضل أسلوب تضعها فيه في برنامجك، زوّد برنامجك بالأفكار فكرة فكرة – بعد أن تجد الأسلوب الأفضل لوضعها ضمن برنامجك – وستجده مع مرور التحديثات أفضل.

البنية التحتية لمكتبة أدواتك الخاصة

آثرت تسمية هذه الفقرة بـ "منصتك الخاصة"، على اعتبار أن المنصات (أطر العمل Frameworks هي بيئات لتطوير التطبيقات توفر أدوات جاهزة (فئات وطرق و ...)، بحيث يمكن للمبرمج إنشاء برمجيات غير محدودة منها (وهذا ما سنقوم به بالفقرات القادمة)، على عكس المكتبات التي تقدم مجموعة من الأكواد الجاهزة على شكل فئات وطرق ولا يمكن للمبرمج تعديل أي سطر برمجي منها. ولكني مع ذلك لم أرد المبالغة وتحميل القارئ آمالًا بما سيراه ويقرأه ثم يتفاجأ بخلاف توقعاته، ولو أن الغاية من هذا الباب واضحة. لذلك، دعنا مبدئيًّا نتفق على أن ما سنقوم به – حتى آخر الباب – هو مكتبة برمجية، فيها أدوات خاصة ومجموعة من الفئات المختلفة.

ماذا عن العجلة (المثل المشهور عن إعادة اختراع الأشياء الموجودة مسبقًا)؟ صحيح أن ما سنقوم به موجود ومستهلَك، وقديمٌ حتى، إلا أنه ليس لغرض إعادة اختراعه باسمنا والتباهي به أمام الناس مثلًا، وإنما لفهم مبدئه، والوصول لأشياء مفيدة غير موجودة مسبقًا (أو موجودة ولكنها مدفوعة أو لا يمكن الوصول إليها لسبب أو لآخر)، فالعملية ليست تكرارًا وتقليدًا بلا غاية أو سبب، وإنما تعليمية بالدرجة الأولى، بالإضافة إلى أنها تقنية وخدمية وقد تكون ربحية. كما أن من ينشئ المنصات والمكتبات (وبشكل مصغّر: التوابع الجاهزة) لا يقال عنه أنه يخترع العجلة من جديد، وإنما هو يقوم باختراع أساس متين لعجلة – جديدة أو قديمة – سيعتمد عليها هو أو غيره والذين قد يصلون للمئات أو الآلاف (أو أكثر من ذلك إذا كان على مستوى الشركات).

بعد أن اطلعتَ على بعض أسرار الأدوات والنصائح المتعلقة بها، وقبل خوض تجربة إنشاء الأدوات، لا بد من ضبط وترتيب بعض الأمور. ستقدم لك هذه الفقرة – بفقراتها الفرعية – مجموعة من التقنيات والخطوات التي يمكنك استخدامها في أكوادك لتنظيمها وتسهيل

صيانتها، بالإضافة لبعض الاعتبارات التي ستجعل مكتباتك أفضل. مع العلم أن هذه الفقرة هي أساس للفقرات التالية، إذ ما سنقوم به هنا هو مجموعة من الفئات ومجموعة من أعضاء الفئات والتي سنعتمد عليها في الفقرات التالية دون تفصيل ولا توضيح.

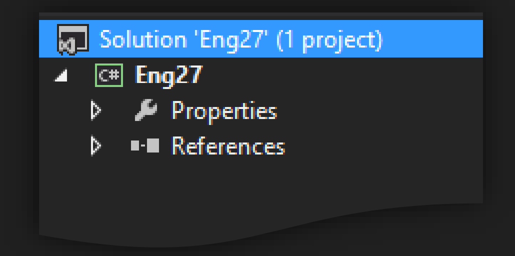
مجال الأسماء، وما فيه

الفقرات السابقة تضمنت أداتين، ركزنا على إحداهما لتطبيق أمثلة تطبيقية للفقرات الفرعية للفقرة السابقة (تقريبًا مثال لكل فقرة). لو لاحظت مجال الأسماء المستخدم فقد كان UserControlsTest، وهو نفسه اسم المشروع، والذي كان يحوي – بجانب فئة الأداتين الخاصتين اللتين أنشأناهما – فئة نافذة المشروع الافتراضية وفئات أخرى.

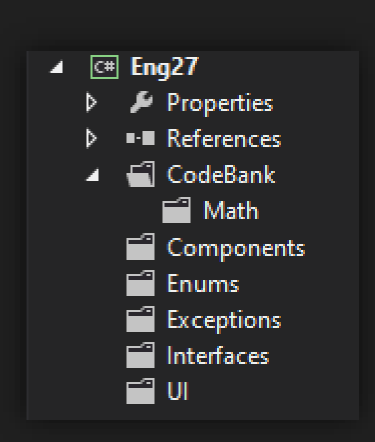
عند تضمين مجال الأسماء الخاص بك – أو ما تسمح به منه – باستخدام الكلمة using ضمن مشروع أحد المبرمجين فإن عليه كتابة اسم مجال الأسماء الحاوي على الفئات التي يرغب باستخدامها أمام الكلمة using. وفي حال كان مجال الأسماء ضمن مجال – أو مجالات – أسماء بشكل متفرع، عليه ذكر المسار كاملًا.

يمكن تشبيه مجالات الأسماء بالمجلدات، والفئات داخله بالملفات، لذلك فإنك لتصل لفئة ما عليك تضمين مجال الأسماء الحاوي عليها، والذي قد يكون فرعيًّا. حتى أن تنظيم فئات المشروع ضمن مجلدات يكافئ إنشاء مجالات أسماء فرعية، ولو أنه لا يعني ذلك بطريقة مباشرة.

كبداية أنشئ مشروعًا من النوع WindowsFormsControlLibrary باسم Eng27 – أو أي اسم ترغب به – لتحصل على فئة UserControl بشكل افتراضي، احذفها ليغدو المشروع فارغًا:



أنشئ مجموعة من المجلدات كما هو موضح بالشكل المجاور. المجلدات التي ستنشئها ما هي إلا مجالات أسماء مكتبة أدواتك. وذلك لإيضاح كيفية التعامل مع مجالات الأسماء المختلفة ومجالات الأسماء الفرعية ضمنها، بهدف تنظيم المشروع وتسهيل العمل عليه.



مجال الأسماء الأول CodeBank سيحوي مجموعة من الأنواع والفئات التي تتضمن مجموعة من الطرق الجاهزة لأغراض محددة سنراها لاحقًا، وضمن مجال الأسماء هذا هناك مجال الأسماء Math والذي لا يختلف عن CodeBank إلا أنه مخصص للفئات الرياضية. مجالات الأسماء Enums وExceptions وInterfaces ستضم الفئات وأعضاء الفئات ذات الصلة باسمها كما هو واضح. أما مجالا الأسماء Components وUI فسيحويان الأدوات الخاصة، ضيوف شرف فصول هذا الباب. كما أن مجال الأسماء الافتراضي Eng27 – خارج المجلدات، ضمن المجلد الجذر – سيحوي بعض الفئات وأنواع البيانات العامة التي يمكن استخدامها في كل فئات المشروع، كما سنرى.

صحيح أن مكونات المكتبة كلما كانت أكثر كانت الفائدة من المكتبة أكبر، لاعتماد المستخدم على المكتبة بشكل أكبر، حتى لو كان هناك تكرار بهذه المكونات (مع مكونات المكتبات والمنصات الأخرى)، وصحيح أنني سأتوسّع وأكثِر وأستزيد من فئات المكتبة ضمن هذا الفصل لتبسيط الفكرة أكثر عن الأدوات الخاصة والمكتبات والمنصات وإيضاح أغلب ما قد يرد على بال القارئ؛ إلا أنني سأقف عند مقدار محدد حتى لو أن هناك أشياء أخرى يمكن إضافتها للمكتبة وإغناءها بها، فإني أرجو أن أنهي كتابي هذا قبل مشيبي.

مواصفات وخصائص المشروع

افتح الفئة AssemblyInfo التي تجدها ضمن مجلد خصائص المشروع Properties واضبط المواصفات فيها بما يناسبك، وافتح خصائص المشروع بالنقر المزدوج على المجلد Properties نفسه ثم انتقل للصفحة Build وفعّل XML documentation file.

الفئات المجردة

عند التعامل مع مصطلحات ومفاهيم لا تمثل شيئًا على أرض الواقع (برمجيًّا لا تشير إلى أي نوع من البيانات) فإننا نصف هذه المصطلحات أو المفاهيم بالمجردة، وفي حالتنا، الفئات المجردة. وكما سبق وذكرنا، فالفئات المجردة لا يمكن استنساخها (إنشاء كائنات منها)، وهذا منطقي جدًّا، إذ لا معنى لهذه الفئات على أرض الواقع وهي مجردة.

يتم تعريف فئة ما على أنها مجرّدة بالكلمة المحجوزة abstract، وعندها لا يمكن الاستفادة من الفئة إلا بالوراثة منها، وهذا منطقي أيضًا، فالمفاهيم المجردة ما هي إلا مفاهيم تضفي على الماديات معان إضافية مشتركة مع ماديات أخرى.

بمعنى آخر، الفئات المجردة هي فئات فيها أعضاء مشتركة بين فئات مختلفة، وعوضًا عن تكرار الأعضاء ضمن كل الفئات التي تحويها، يمكن – باستخدام الفئات المجرّدة – وضع الأعضاء المشتركة ضمن فئة – هي الفئة المجردة – ووراثتها.

ماذا عن الواجهات Interfaces؟

قد يبدو مفهوم الواجهات غريبًا للوهلة الأولى، لكنك عندما تعتاد عليه وتفهمه سترى كل شيء في حياتك عبارة عن أصفار وواحدات 😜 (كما حصل معك عندما أنجزت برنامجك الأول، والذي غالبا ما كان "Hello World"، لكن هذه المرة جد).

ذكرنا أن الفئات المجردة هي فئات فيها أعضاء مشتركة بين فئات مختلفة، نستخدمها لعدم تكرار هذه الأعضاء ضمن كل الفئات التي ترث من هذه الفئات المجردة. وبشكل مشابه – بطريقة أو بأخرى – تستخدم الواجهات لذات الهدف.

في حال كنت قد تعرفت على مفهوم الواجهات من قبل فمن المؤكد أنك سمعت أن الغاية من الواجهات هو الوراثة المتعددة، بمعنى أن ترث الفئة من أكثر من مصدر. فهل هذا صحيح؟

في الواقع، هذا صحيح، ولكن ليس بالمعنى الحرفي. فعندما ترث فئة من فئة وواجهة أو أكثر (وراثة متعددة)، فإنها ترث كل ما تملكه الفئة الأم، في حين أنها لا ترث إلا أسماء

أعضاء الواجهات التي ترث منها، إذا إن الواجهات لا تحوي إلا أسماء أعضائها! وبكلام أدق: هي ترث أعضاء الفئة الأم، وتوظف أسماء أعضاء الواجهات.

أضف إلى ذلك، أن الواجهات تفرّق الفئات وتميزها، وتعطيك إمكانية المفاضلة بين الفئات والتعامل مع كل فئة بشكل مختلف عن الأخرى، بحسب الواجهات التي تعود إليها.

إذا شبّهنا الناس بالكائنات Objects، يمكنك أن تقول عندها أنهم من النوع إنسان Human، أو بلغة برمجية: من الفئة Class إنسان، إذ إنها تصف شيئًا عامًّا. كما يمكنك القول أن هذه الفئة مشتقة من فئة مجردة لا وجود لها على أرض الواقع: فئة الكائن الحي، وهي فئة مجردة تمثل مفهوما ما تشترك فيه أكثر من فئة، كالبشر والحيوانات والنباتات في مثالنا (كلها فئات لها وجود على أرض الواقع، وتشترك فيما بينها بالفئة المجردة فئة الكائن الحي، فكل هذه الفئات تقوم على المبدأ ذاته: تحيا، وتموت، وتأكل، وتتكاثر، لها أجهزة عصبية وهضمية وغيرها، وتمر عليها ظروف مثل التعب والجوع والمرض و...). أما بالنسبة للواجهات، فيمكنك القول أن الكائنات – في مثالنا السابق – تتميز عن بعضها بالعمل مثلا، فالبشر الذين يعملون في التعليم لهم ما يميزهم عمن يعملون في الجيش، عمن يعملون في البنوك، عمن يعملون في المشافي، عن غيرهم من الناس! وعلى ذلك فقس.

والآن.. أنشئ واجهة باسم IEng27Control ضمن مجال الأسماء Eng27.Interfaces، واحذف جميع مجالات الأسماء المصرَّح عنها بالكلمة using غير الضرورية؛ ليبدو الكود أسهل:



```
using Eng27.Components;
using Eng27.Enums;

namespace Eng27.Interfaces
{
    /// <summary>
    /// Specifies that this object is Eng27Control.
    /// </summary>
    public interface IEng27Control
    {
        Style_e Style { get; set; }

        StyleDark StyleDark { get; set; }
        StyleLight StyleLight { get; set; }
        ThemeManager ThemeManager { get; set; }
        Theming_e Theming { get; set; } } }
```

هذه الواجهة ستعطي الأدوات التي سترث منها مجموعة من الخصائص، وهي ما ستميزها عن غيرها من الأدوات.

أنشئ واجهة أخرى فارغة باسم IEng27Component لتمثل مكونات المكتبة.



هناك عرف بين المستخدمين على إضافة الحرف I إلى بداية أسماء الواجهات لتمييزها عن الفئات، يحبّذ أن تتبعه.
حتى لو كانت الواجهة فارغة، يبقى لها أهمية لتمييز الفئات عن بعضها.



يمكن إنشاء واجهة Interface من خلال Project > Add New Item، أو بإضافة فئة وتعديل تعريفها، أو بكتابة الواجهة وما تحويه ضمن ملف فئة أو واجهة أخرى، أو بإنشاء ملف فارغ ضمن المشروع.

الواجهة الأخيرة فيها بضعة أعضاء، معددين enum وثلاثة فئات. أنشئ المعددين داخل مجال الأسماء Eng27.Enums، أما الفئات فأنشئها في مجال الأسماء Eng27 مباشرة:

```
namespace Eng27.Enums
{
    /// <summary>
    /// Defines a Style enumeration.
    /// </summary>
    public enum Style_e
    {
        Dark,
        Light
    }
}
```



```
using System.Drawing;

namespace Eng27
{
    /// <summary>
    /// Represents the Light style of Eng27 controls.
    /// </summary>
    public class StyleLight : Style
    {
        /// <summary>
        /// Creates a new instance of StyleLight class.
        /// </summary>
        public StyleLight()
        {
            this.BackColor = Color.DodgerBlue;
            this.ForeColor = Color.FromArgb(222, 222, 222);
            this.Font = new Font("Segoe UI Light", 8.25F);
        }
    }
}
```



```
using System.Drawing;

namespace Eng27
{
    /// <summary>
    /// Represents the Dark style of Eng27 controls.
    /// </summary>
    public class StyleDark : Style
    {
        /// <summary>
        /// Creates a new instance of StyleDark class.
        /// </summary>
        public StyleDark()
        {
            this.BackColor = Color.FromArgb(33, 33, 33);
            this.ForeColor = Color.FromArgb(222, 222, 222);
            this.Font = new Font("Segoe UI Light", 8.25F);
        }
    }
}
```

كما هو واضح، فالفئتين الأخيرتين ترثان من فئة اسمها Style، وهذا كودها:



```
using System;
using System.ComponentModel;
using System.Drawing;

namespace Eng27
{
    /// <summary>
    /// Represents the Style of Eng27 controls.
    /// </summary>
    [RefreshProperties(RefreshProperties.Repaint)]
    public class Style
```

```
    {
        #region Constructors
        /// <summary>
        /// Creates a new instance of Style class.
        /// </summary>
        public Style() { }
        #endregion


        #region Properties
        private Color backcolor;
        /// <summary>
        /// Gets or sets the background color for the control.
        /// </summary>
        [Description("Gets or sets the background color for the control.")]
        public Color BackColor
        {
            get { return backcolor; }
            set { backcolor = value; this.OnStyleChanged(new EventArgs()); }
        }

        private Color forecolor;
        /// <summary>
        /// Gets or sets the foreground color for the control.
        /// </summary>
        [Description("Gets or sets the foreground color for the control.")]
        public Color ForeColor
        {
            get { return forecolor; }
            set { forecolor = value; this.OnStyleChanged(new EventArgs()); }
        }

        private Font font;
        /// <summary>
        /// Gets or sets the font of the text displayed by the control.
        /// </summary>
        [Description("Gets or sets font of the text displayed by the control.")]
        public Font Font
        {
            get { return font; }
            set { font = value; this.OnStyleChanged(new EventArgs()); }
        }
        #endregion


        #region Methods
        /// <summary>
        /// Returns a string that represents the style.
        /// </summary>
        public override string ToString()
        {
            return string.Format(
                "{0}, {1}pt. Colors: {2} - {3}",
                font.Name,
                font.Size,
                backcolor.Name,
                forecolor.Name);
        }
        #endregion
```

```
        #region Custom Events
        /// <summary>
        /// Occurs when Style changed.
        /// </summary>
        public event EventHandler StyleChanged;
        /// <summary>
        /// Raises the Eng27.StyleChanged event.
        /// </summary>
        /// <param name="e">
        /// An System.EventArgs that contains the event data.
        /// </param>
        protected virtual void OnStyleChanged(EventArgs e)
        {
            if (StyleChanged != null)
                StyleChanged.Invoke(this, e);
        }
        #endregion
    }
}
```

ستحتاج لإضافة مجال الأسماء System.Drawing إلى مراجع المشروع. وقبل أن نشرح الفئات السابقة، سنؤجل الحديث عن الفئة ThemeManager لفقرة لاحقة.

والغاية من الفئات الثلاث السابقة تزويد الأدوات بنمطين: المظلم و المضيء، وحتى لا نكرر الأكواد في الفئتين، أنشأنا فئةً تمثل مفهوم النمط أو المظهر Style، وأنشأنا فئتين ترثان منها، بحيث نضبط خصائص كل فئة على حدة.

وصّفنا الفئة بالمواصفة RefreshProperties، والذي يجعل صندوق الخصائص يحدّث نفسه، إذ إن هذه الفئة هي خاصية فيها خصائص فرعية (جميع الفئات التي سترث من هذه الفئة ستتصف بهذه المواصفة).

أعدنا تعريف الطريقة ToString بحيث نحدد ما سيتم إعادته عند استدعاء التابع، وما سيظهر في صندوق الخصائص أمام الخاصية التي مصدرها هذه الفئة (إذا لم نقم بهذا فستكون القيمة النصية لكائنات هذه الفئة قيمة فارغة).



الخصائص ما هي إلا كائنات مستنسخة من فئات ما.

كيف يمكن الاستفادة من الواجهة؟

بالإضافة لإجبار الفئات التي ترث من الواجهة على احتواء مجموعة من الإعضاء، فإنه – من خلال الواجهات – يمكنك اختصار الكثير من الأكواد التي تحتاجها للوصول لنتيجة ما عندما يكون لديك الكثير من الفئات التي تحتاج لتنفيذ أوامر مختلفة تبعا للفئة بعينها.

لو كانت لديك مجموعة من الفئات (مثلا 10 فئات)، وكلها تعود لمصدر واحد (فئة أو مجموعة من الفئات ترث منها بشكل متسلسل)، وكانت بعض هذه الفئات لها اعتبار معين، والبعض الآخر له اعتبار آخر، وغيرها له اعتبار غيره، فمن غير المنطقي أن تكتب هذا:

```
if (obj is Class1 || obj is Class2 || ...)
    DoSomething();
else if (obj is Class5 || obj is Class6 || ...)
    DoSomething2();
.
.
```

وعوضا عن ذلك:

```
if (obj is IMyInterface1)
    DoSomething();
else if (obj is IMyInterface2)
    DoSomething2();
.
.
```

ومعنى الكود الأخير: إذا كان الكائن يعود للواجهة الفلانية فافعل كذا، وإذا كان يعود لواجهة غيرها فافعل كذا. في حين أنه في الكود الأول كان عليك كتابة جميع الفئات التي يمكن أن يَكُونها الكائن لتنفيذ الأمر الفلاني، وكذا للشروط الأخرى. كما ينبغي عليك في هذه الحالة تعديل الكود كلما أضفت أو أزلت أو عدلت فئات مشروعك! بينما لا حاجة لك بذلك لو كنت تتعامل مع الواجهات، يكفي وراثة فئة لواجهة ما وستصبح جميع كائنات هذه الفئة لها صلة قرابة مع هذه الواجهة!

إن لم تصل لتصور جيد عن مفهوم الواجهات، فالأدوات التي سنشرحها في الفقرات التالية ستجعل الأمر أسهل؛ لذلك فأكمل القراءة، فالقادم أسهل وأجمل.

أحداثك الخاصة Custom Events

توفر لك الأدوات التي سترث منها أدواتُك الكثيرَ من الأحداث القياسية، لكنك قد تحتاج لتأليف أحداث خاصة بك، تحصل في ظروف خاصة لا تدعمها أو توفرها الأحداث القياسية. كما أنك تحتاج لإنشاء أحداث خاصة عند تأليف أدوات هجينة (مختلطة، مركبة من أكثر من أداة) أو عند تأليف أدوات من الصفر.

تحتاج الأحداث كائنين لتعلم على من وقعت، والحيثيات التي تمثل الحدث الحادث. الأول كائن من الفئة Object يمثل مَن وقع عليه الحدث، والثاني نسخة عن الفئة EventArgs – أو ما يرث منها – يمثل ما حدث.

الحدث المعرّف من خلال الفئة EventArgs:

لعل ما يدور في ذهنك الآن هو المفوّض delegate، والذي لطالما ارتبط ذكره بالأحداث Events. لكنك لن تحتاجه إذا اعتمدت على الفئة EventArgs، إذ إنها تغنيك عنه.

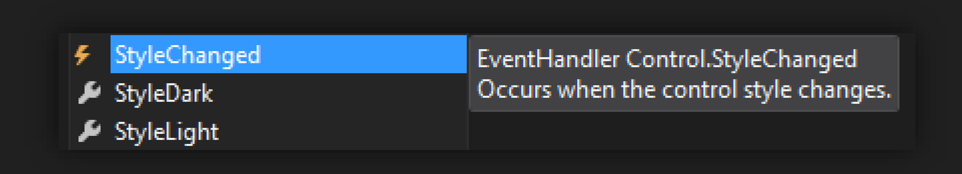
أنشأنا حدثًا خاصًّا يتم تفجيره عند تغيير النمط Style، هذا ما يعرفه المبرمج ويهمه ويعنيه، أما نحن كمؤلفين فيعنينا أكثر من ذلك، فعلينا تحديد متى يتم تفجير الحدث (متى يحدث الحدث)، وأظن أن "this.OnStyleChanged(new EventArgs());" قد لفت انتباهك، فهو الذي يحدد متى ينفجر الحدث!

وكملخص للأحداث: لإنشاء حدث خاص بك تحتاج:

- مفوّض Delegate، وفي هذه الفئة اعتمدنا على مفوّض قياسيّ.
- إجراء (نبدؤه بـ On، ونسميه باسم المفوّض الذي أنشأناه).
- استدعاء الإجراء في الأماكن المتعلقة بالحدث.



المفوّض مسؤول عن ربط الحدث بالفئة، وذلك بإعطاء الحدث البيانات التي تصف الحدث EventArgs لاحظ:



الحدث المعرّف من خلال فئة مشتقة من EventArgs:

إذا كنت تود تمرير تفاصيل ضمن الحدث عليك إنشاء فئة – مشتقة من الفئة EventArgs – تمثل حيثيات الحدث، وهنا دور المفوض لربط الحدث بالطريقة التي ستمثل الحدث (إذ إن الأحداث ليست إلا طرقًا مُدارة من قبل مندوبين delegates). الأداة ThemeManager فيها حدث معرف بهذا الأسلوب، ستراها لاحقًا.

متى يتم تفجير الأحداث؟

في أي مكان من الفئة يتم استدعاء الإجراء المرتبط بالحدث يؤدي لتفجيره، سترى ذلك بوضوح لاحقًا.

فئات الأدوات الأساس (الأم) BaseControls

كما تعلم فإن الغاية من الطرق (التوابع والإجراءات) هي توفير الوقت وتنظيم الكود، فلولاها كان عليك تكرار مئات الأسطر البرمجية مرات ومرات، هذا فضلًا عن تعديلها وصيانتها! وبشكل مشابه، فإن البرمجة الكائنية تقوم بشيء مماثل.

ذكرنا في بداية هذا الفصل أن بعض الأدوات تشترك فيما بينها بمجموعة من الأعضاء، لذلك فإننا سننشئ فئة مشتركة بين فئات هذه الأدوات، وهي ما تسمى عادة بفئة الأساس (مثلًا أدوات الأزرار فئتها الأساس ButtonBase). إن كانت الفقرات السابقة واضحة بالنسبة لك، فمن المرجح أنه قد خطر ببالك أن الفئات الأساس هي فئات مجردة.

وكمثال قبل أن نبدأ، اكتب سطرًا برمجيًا يحوي الفئة Button، وانقل مؤشر الفأرة إليها وانقر على F12 (Go To Definition)، لاحظ أن هذه الفئة مشتقة من فئة أخرى وواجهة، وأن عدد أكوادها هو 145:

```
1   Assembly System.Windows.Forms.dll, v4.0.0.0
4
5   using System;
6   using System.ComponentModel;
7   using System.Runtime.InteropServices;
8
9   namespace System.Windows.Forms
10  {
11      ...public class Button : ButtonBase, IButtonControl
17      {
18          ...public Button();
21
22          ...public AutoSizeMode AutoSizeMode { get; set; }
32          ...protected override CreateParams CreateParams { get; }
41          ...public virtual DialogResult DialogResult { get; set; }
54
55          ...public event EventHandler DoubleClick;
60          ...public event MouseEventHandler MouseDoubleClick;
67
68          ...public virtual void NotifyDefault(bool value);
77          ...protected override void OnClick(EventArgs e);
83          ...protected override void OnFontChanged(EventArgs e);
91          ...protected override void OnMouseEnter(EventArgs e);
97          ...protected override void OnMouseLeave(EventArgs e);
103         ...protected override void OnMouseUp(MouseEventArgs mevent);
112         ...protected override void OnTextChanged(EventArgs e);
120         ...public void PerformClick();
124         ...protected internal override bool ProcessMnemonic(char charCode);
135         public override string ToString();
136         ...protected override void WndProc(ref Message m);
144     }
145 }
```



عدد الأسطر هو فقط عدد أسطر التوثيقات والتوصيفات، فالأكواد البرمجية المكونة لأعضاء الفئات التي نستعرضها من خلال النقر على F12 في حال كانت المكتبة التي تستعرض فئاتها عبارة عن مكتبة dll؛ لا يمكنك معرفة شيء عنها حتى عددها.

ولو نقلت المؤشر للواجهة IButtonControl لوجدت:

```
1  ⊞ Assembly System.Windows.Forms.dll, v4.0.0.0
4
5    using System;
6
7  ⊟ namespace System.Windows.Forms
8    {
9  ⊞     ...public interface IButtonControl
12       {
13 ⊞         ...DialogResult DialogResult { get; set; }
19
20 ⊞         ...void NotifyDefault(bool value);
28 ⊞         ...void PerformClick();
32       }
33   }
```

بينما لو نقلته لفئة ButtonBase لوجدتها مشتقة من الفئة Control، وأنها فئة مجردة، وأن عدد أكوادها 428، وفيها طرق وخصائص وأحداث أكثر.

```
13 ⊟     ...public abstract class ButtonBase : Control
```

ولو نقلت المؤشر للفئة Control ونقرت على F12 لوجدتها مشتقة من مصادر عديدة وأن عدد أكوادها يتجاوز 3000!

```
17 ⊟     ...public class Control : Component, IDropTarget, ISynchronizeInvoke, IWin32Window, IBindableComponent, IComponent, IDisposable
```

وهكذا إلى أن تصل إلى مصدرٍ لا مصدرَ له، أو بكلام آخر: إلى موَرِثٍ لا موَرِّثَ له.

يمكنك الحصول على رؤية أوضح لأصول وجذور فئة ما من خلال عرض الفئة Class View:

```
◢ Button
    ◢ ButtonBase
        ◢ Control
            ◢ Component
                ◢ IComponent
                    IDisposable
                  IDisposable
                ◢ MarshalByRefObject
                    Object
            ◢ IBindableComponent
                ◢ IComponent
                    IDisposable
                  IDisposable
            ◢ IComponent
                  IDisposable
              IDisposable
              IDropTarget
              ISynchronizeInvoke
              IWin32Window
      IButtonControl
```



يمكن الوصول لعرض الفئة Class View من خلال لوحة المفاتيح من خلال الاختصار Ctrl + Shift + C أو من خلال القائمة View > Class View.

والآن.. أنشئ فئة سمّها Eng27ButtonBase واجعلها مجردة abstract ودعها ترث من الفئة Button ومن الواجهة IEng27Control:

```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System.ComponentModel;
using System.Windows.Forms;

namespace Eng27
{
    /// <summary>
    /// Implements the basic functionality common to eng27 button controls.
    /// </summary>
    public abstract class Eng27ButtonBase : Button, IEng27Control
    {
```

```
        #region Constructors
        /// <summary>
        /// Initializes a new instance of the Eng27.Eng27ButtonBase class.
        /// </summary>
        public Eng27ButtonBase()
        {
            this.FlatStyle = FlatStyle.Flat;
            this.FlatAppearance.BorderSize = 0;
        }
        #endregion


        #region Properties
        private Style_e style;
        public Style_e Style
        {
            get { return style; }
            set { style = value; }
        }

        StyleDark styledark = new StyleDark();
        [Browsable(true)]
        [TypeConverter(typeof(ExpandableObjectConverter))]
        [RefreshProperties(RefreshProperties.Repaint)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Content)]
        [Description("StyleDark of Control.")]
        public StyleDark StyleDark
        {
            get { return styledark; }
            set { styledark = value; }
        }

        StyleLight stylelight = new StyleLight();
        [Browsable(true)]
        [TypeConverter(typeof(ExpandableObjectConverter))]
        [RefreshProperties(RefreshProperties.Repaint)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Content)]
        [Description("StyleLight of Control.")]
        public StyleLight StyleLight
        {
            get { return stylelight; }
            set { stylelight = value; }
        }

        private Theming_e theming = Theming_e.AllowFull;
        public Theming_e Theming
        {
            get { return theming; }
            set { theming = value; }
        }

        public ThemeManager ThemeManager { get; set; }
        #endregion
    }
}
```

لاحظ الوراثة من الواجهة IEng27Control، جميع الأدوات التي سترث من هذه الفئة سترتدي سمات تلك الواجهة. لاحظ أيضًا أن الأداة في البداية تقوم بضبط بعض الخصائص، وعلى اعتبارها تابعة للواجهة IEng27Control فعليها أن تحوي الأعضاء التي تحويها الواجهة. سنناقش هذه الخصائص في القادم من الأدوات.

ستحتاج لإضافة المكتبة System.Windows.Forms إلى مصادر مشروعك.



لا تنس جعل فئاتك عامة، حتى يتسنى للمبرمجين الوصول إليها من مشاريعهم.
حاول تجنب استخدام أسماء الفئات المستخدمة لألا يحصل لبس، ولتوفر على المبرمجين عناء كتابة مجال الأسماء التفصيلي الحاوي على الفئة التي حصل فيها هذا اللبس.

وعلى سبيل كائنية التوجه، سننشئ فئة سنسميها Eng27Control، سننشئ منها أدواتنا إذا لم تكن هناك أداة قياسية توصلنا لغايتنا بتعديل خصائصها:



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System.ComponentModel;
using System.Windows.Forms;

namespace Eng27
{
    ///
    public class Eng27Control : Control, IEng27Control
    {
        #region Constructors
        public Eng27Control() { }
        #endregion


        #region Properties
        private ThemeManager thememanager;
        public ThemeManager ThemeManager {
            get { return thememanager; }
            set
            {
                thememanager = value;
                if (thememanager != null)
                    thememanager.ThemeChanged += thememanager_ThemeChanged;
            }
        }
```

```
        private Style_e style;
        public Style_e Style
        {
            get { return style; }
            set { style = value; this.ChangeStyle(); }
        }

        StyleDark styledark = new StyleDark();
        [Browsable(true)]
        [TypeConverter(typeof(ExpandableObjectConverter))]
        [RefreshProperties(RefreshProperties.Repaint)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Content)]
        [Description("StyleDark of Control.")]
        public StyleDark StyleDark
        {
            get { return styledark; }
            set { styledark = value; }
        }

        StyleLight stylelight = new StyleLight();
        [Browsable(true)]
        [TypeConverter(typeof(ExpandableObjectConverter))]
        [RefreshProperties(RefreshProperties.Repaint)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Content)]
        [Description("StyleLight of Control.")]
        public StyleLight StyleLight
        {
            get { return stylelight; }
            set { stylelight = value; }
        }

        Theming_e theming;
        public Theming_e Theming
        {
            get { return theming; }
            set { theming = value; }
        }
        #endregion


        #region Methods
        public virtual void ChangeStyle() {
            switch (style)
            {
                case Style_e.Dark:
                    this.BackColor = styledark.BackColor;
                    this.ForeColor = styledark.ForeColor;
                    this.Font = styledark.Font;
                    break;

                case Style_e.Light:
                    this.BackColor = stylelight.BackColor;
                    this.ForeColor = stylelight.ForeColor;
                    this.Font = stylelight.Font;
                    break;

                default: break;
            }
        }
        #endregion
```

```
        #region Events
        void StyleLight_StyleChanged(object sender, System.EventArgs e)
        {
            this.ChangeStyle();
        }

        void StyleDark_StyleChanged(object sender, System.EventArgs e)
        {
            this.ChangeStyle();
        }

        private void thememanager_ThemeChanged(object sender, ThemingEventArgs e)
        {
            switch (e.Theming)
            {
                case ThemingArgs_e.StyleChanged:
                    if (this.Theming == Theming_e.AllowBorderOnly ||
                        this.Theming == Theming_e.Disallow)
                        break;

                    styledark = thememanager.StyleDark;
                    stylelight = thememanager.StyleLight;
                    style = thememanager.Style;
                    this.ChangeStyle();
                    break;

                case ThemingArgs_e.BorderSizeChanged:
                    break;

                case ThemingArgs_e.BorderColorChanged:
                    break;

                default: break;
            }
        }
        #endregion
    }
}
```

الأداة Eng27Control هي نفسها الأداة Control القياسية، لكنها تحوي بعض الخصائص الإضافية التي اكتسبتها من الواجهة IEng27Control، كما أن فيها إجراءً يغير النمط الخاص بالأداة. (الفئتان الأنماط Styles ومدير السمات ThemeManager سنناقشها لاحقًا).

أدوات إدارة التصميم في وقت التنفيذ Run-Time

بعض المنصات تعتمد على أدوات لإدارة مجموعة من الأدوات الأخرى، فعوضًا عن التحكم بكل أداة بأداتها – خصوصًا إن كان هذا التحكم نفسه لكل الأدوات – يمكنك إنشاء أداة تتحكم بجميع الأدوات وفق منحى معين. هذه الأداة هي ThemeManager، وستُعنى بمظهر الأدوات (سنكتفي بلون الخط ونوعه ولون الخلفية ولون الحواف وحجمها). ما ستقوم به هذه الأداة هو تخزين بيانات على شكل قالب ما، بحيث كلما تغير هذا القالب تتغير الأدوات المرتبطة بهذه الأداة. كما لا حاجة لكون الأداة ذات طبيعة رسومية، لذلك سنجعلها مكونًا Component. قبل أن نناقش الأداة سنسرد ما نحتاجه، لنرسم خريطةً نسير عليها للوصول للأداة المنشودة. ما نحتاجه:

- أداة ذات طبيعة غير رسومية (مكوّن Component) فيها مجموعة من الخصائص.
- حدث يخبرنا عندما يتم تغيير بيانات هذه الأداة.
- لمزيد من التفصيل، ولتكون الفائدة أكبر، على الحدث أن يشمل على نوع البيانات التي سيتم تغييرها (هل تم تغيير النمط Style؟ أم حجم الحواف؟ أم لون الحواف؟)؛ وبالتالي فإننا نحتاج معدّدا يحوي ما يمكن للبيانات أن تتغير وفقه.
- في الأدوات التي ستُدار من قبل هذه الأداة، نحتاج خاصية تشمل على هذه الأداة (إذا لم يتم ضبطها فإن الأداة لا يتم إدارتها).
- لمزيد من التفصيل أيضًا، سننشئ خاصية في الأدوات التي سيتم التحكم بها من قبل هذه الأداة تحوي ما يمكن لهذه الأداة التحكم به؛ أي أننا سنحتاج معدّدا يحوي ما يمكن لهذه الأداة أن تقوم به في الأدوات المدارة.

```
namespace Eng27.Enums
{
    ///
    public enum Theming_e
    {
        AllowFull = 0,
        AllowStyleOnly = 1,
        AllowBorderOnly = 2,
        Disallow = 3
    }
}
```



```
namespace Eng27.Enums
{
    ///
    public enum ThemingArgs_e
    {
        StyleChanged,
        BorderSizeChanged,
        BorderColorChanged
    }
}
```



```
using Eng27.Enums;
using System;

namespace Eng27
{
    ///
    public class ThemingEventArgs : EventArgs
    {
        ///
        public ThemingEventArgs(ThemingArgs_e e) {
            theming = e;
        }

        private ThemingArgs_e theming;
        ///
        public ThemingArgs_e Theming
        {
            get { return theming; }
        }
    }
}
```



```
using Eng27.Enums;
using Eng27.Interfaces;
using System.ComponentModel;
using System.Drawing;

namespace Eng27.Components
{
    /// <summary>
    /// Manages style of eng27 controls.
    /// </summary>
    [DefaultEvent("ThemeChanged")]
    [DefaultProperty("Style")]
    [DesignerCategory("Eng27 Components")]
    public partial class ThemeManager : Component, IEng27Component
    {
        #region Constructors
        /// ...
        public ThemeManager()
        {
            InitializeComponent();
            styledark.StyleChanged += styledark_StyleChanged;
            stylelight.StyleChanged += stylelight_StyleChanged;
        }
```

```
        /// ...
        public ThemeManager(IContainer container)
        {
            container.Add(this);

            InitializeComponent();
            styledark.StyleChanged += styledark_StyleChanged;
            stylelight.StyleChanged += stylelight_StyleChanged;
        }
        #endregion


        #region Properties
        private Style_e style = Style_e.Light;
        /// <summary>
        /// Gets or sets the Style of controls managed by this component.
        /// </summary>
        [Category("Eng27")]
        [DefaultValue(typeof(Style_e),"Light")]
        [Description("Gets or sets the Style of controls managed by component.")]
        public Style_e Style
        {
            get { return style; }
            set
            {
                style = value;
                this.OnThemeChanged(
                    new ThemingEventArgs(
                        ThemingArgs_e.StyleChanged));
            }
        }

        StyleDark styledark = new StyleDark();
        /// <summary>
        /// Gets or sets the StyleDark of controls managed by this component.
        /// </summary>
        [Category("Eng27")]
        [Description("Gets or sets the StyleDark of controls managed by...")]
        [Browsable(true)]
        [TypeConverter(typeof(ExpandableObjectConverter))]
        [RefreshProperties(RefreshProperties.Repaint)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Content)]
        public StyleDark StyleDark
        {
            get { return styledark; }
            set { styledark = value; }
        }

        StyleLight stylelight = new StyleLight();
        /// <summary>
        /// Gets or sets the StyleLight of controls managed by this component.
        /// </summary>
        [Category("Eng27")]
        [Description("Gets or sets the StyleLight of controls managed by...")]
        [Browsable(true)]
        [TypeConverter(typeof(ExpandableObjectConverter))]
        [RefreshProperties(RefreshProperties.Repaint)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Content)]
        public StyleLight StyleLight
        {
```

```
            get { return stylelight; }
            set { stylelight = value; }
        }

        private int bordersize = 1;
        /// <summary>
        /// Gets or sets the BorderSize of controls managed by this component.
        /// </summary>
        [Category("Eng27")]
        [DefaultValue(1)]
        [Description("Gets or sets the BorderSize of controls managed by...")]
        public int BorderSize
        {
            get { return bordersize; }
            set
            {
                bordersize = value;
                this.OnThemeChanged(
                    new ThemingEventArgs(
                        ThemingArgs_e.BorderSizeChanged));
            }
        }

        private Color bordercolor = Color.Black;
        /// <summary>
        /// Gets or sets the BorderColor of controls managed by this component.
        /// </summary>
        [Category("Eng27")]
        [DefaultValue(typeof(Color), "Black")]
        [Description("Gets or sets the BorderColor of controls managed by...")]
        public Color BorderColor
        {
            get { return bordercolor; }
            set
            {
                bordercolor = value;
                this.OnThemeChanged(
                    new ThemingEventArgs(
                        ThemingArgs_e.BorderColorChanged));
            }
        }

        private bool use_border_settings = false;
        ///
        public bool UserBorderSettings
        {
            get { return use_border_settings; }
            set { use_border_settings = value; }
        }
        #endregion


        #region Events
        void stylelight_StyleChanged(object sender, System.EventArgs e)
        {
            this.OnThemeChanged(new ThemingEventArgs(ThemingArgs_e.StyleChanged));
        }
```

```
        void styledark_StyleChanged(object sender, System.EventArgs e)
        {
            this.OnThemeChanged(new ThemingEventArgs(ThemingArgs_e.StyleChanged));
        }
        #endregion


        #region Custom Events
        public delegate void ThemingEventHandler(object sen, ThemingEventArgs e);
        /// <summary>
        /// Occurs when the theme changed.
        /// </summary>
        [Category("Eng27")]
        [Description("Occurs when the theme changed.")]
        public event ThemingEventHandler ThemeChanged;
        /// <summary>
        /// Rises the Eng27.Components.ThemeManager.ThemeChanged event.
        /// </summary>
        /// <param name="e">
        /// An Eng27.ThemingEventArgs that contains the event data.
        /// </param>
        protected virtual void OnThemeChanged(ThemingEventArgs e)
        {
            if (ThemeChanged != null)
                ThemeChanged.Invoke(this, e);
        }
        #endregion
    }
}
```

عند إنشاء نسخة من هذه الفئة، يتم ربط الخصائص StyleLight وStyleDark بالحدث StyleChanged، والذي عندما يتم تفجيره يستدعي الإجراء OnThemeChanged مما يؤدي لتفجيره.

الخاصية Style تضبط نمط الأداة، المظلم أم المضيء، وتستدعي إجراء تغيير السمة.

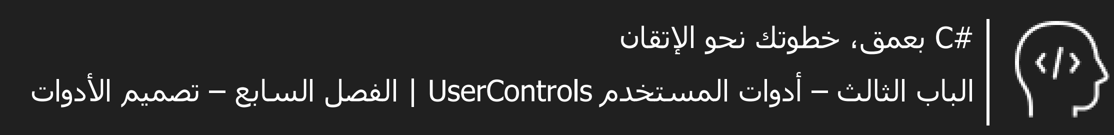
هناك أربع وضعيات لتحكم أداة إدارة السمات ThemeManager بالأدوات: السماح بالتحكم الكامل، التحكم بالنمط Style فقط، التحكم بالحواف فقط، عدم السماح بالتحكم.

الحدث ThemeChanged غير معرف بالمفوّض القياسي EventHandler، وإنما بمفوّض خاص أسميناه ThemingEventHandler، والذي يأخذ بيانات من النوع ThemingEventArgs، والتي لا تحوي فيها إلا كائنًا من النوع ThemingArgs_e.



في الحقيقة، لا نحتاج لإنشاء حدث من مفوض خاص إلا إذا كنت ترغب بتضمين بيانات معينة مع الحدث، فإذا كان الحدث الذي ترغب بإنشاءه لا يتطلب تمرير بيانات معه، فاستخدم المفوّض EventHandler ولا توجّع رأسك.

على سبيل المثال، البيانات التي سيمررها هذا المفوّض للحدث عند تفجيره هي معدد يعبر عن ماذا تم التحكم به، وهنا لي غايتان: إيضاح الفكرة، والاستفادة من هذه البيانات التي تم تمريرها بحد ذاتها، فعلى أساسها يتم ضبط مظهر الأدوات. عد إلى كود الأداة Eng27Control، في نهايته الإجراء المتعلق بالحدث ThemeChanged، فيه تم التحقق بما تم تغييره من خلال مدير السمات ThemeManager، فإذا كانت الأداة تقبل هذا التغيير – بضبط خاصية Theming فيها على ما ستسمح بتغييره – فإنه يتم التغيير، وإلا لا يحدث شيء.



الفصل السابع – تصميم الأدوات

الأدوات التي سنؤلفها في فقرات هذا الفصل ستتنوع بين أساليب إنشاء الأدوات التي ذكرناها في الفصل السادس، ولكني لن أقسّم هذه الفقرات على هذا الأساس، فلن أبدأ بفقرة اسمها: تعديل أدوات موجودة مسبقًا، ثم فقرة: دمج الأدوات، ثم فقرة: رسم الأدوات من الصفر. وإنما سأعتمد معايير أخرى.

أغلب أكواد الأدوات غير مشروحة بالتفصيل، ومعظمها مكونة من مئات الأسطر البرمجية، فحاول فهم الأكواد قبل نقلها، فأي خطأ أو نقص قد يعطي نتائج خاطئة.

هذا بالنسبة لمحتوى الكتاب، أما بالنسبة لمحتوى المشروع في الفيجوال ستوديو فسيكون كالتالي:

ضمن مجال الأسماء Eng27 مباشرة، هناك الفئات والأدوات العامة:

```
{ } Eng27
    ButtonDoubleAnimatedEventArgs
    ChartData
    DataModel
    Eng27ButtonBase
    Eng27Control
    Eng27UserControl
    GradiantColors
    PaintRectangle
    Style
    StyleDark
    StyleLight
    ThemingEventArgs
    ToggleSwitchColors
```

ضمن مجال الأسماء Eng27.CodeBank، هناك بعض الفئات التي تعطي المبرمج وظائف مفيدة:

```
{ } Eng27.CodeBank
    Enums
    ExtenssionMethods
    Transparency
```

الفئة Eng27.CodeBank.Math فيها الفئات الرياضية:

```
◢ { } Eng27.CodeBank.Math
    ▷ Acceleration
    ▷ AccelerationUnits_e
    ▷ Angle
    ▷ AngleType_e
    ▷ ComplexNumber
    ▷ Constants
    ▷ Distance
    ▷ DistanceUnits_e
    ▷ MathObject
    ▷ PhysicalObject
    ▷ PolarComplexNumber
    ▷ PrecisionException
    ▷ RectangularComplexNumber
    ▷ Rounding
    ▷ Speed
    ▷ SpeedUnits_e
    ▷ Time
    ▷ TimeUnits_e
```

مجال الأسماء Eng27.Components يحوي مكوّنات المكتبة (الأدوات غير الرسومية):

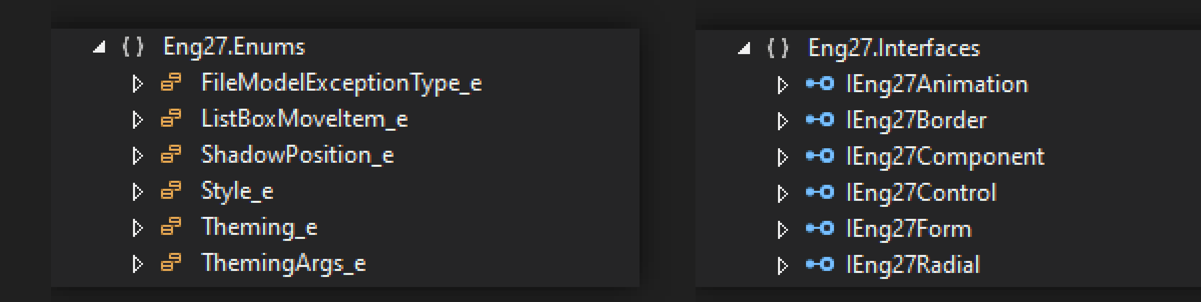
```
◢ { } Eng27.Components
    ▷ ColorTransition
    ▷ ControlDrag
    ▷ ControlExtensionProperty
    ▷ ControlRounding
    ▷ ControlScreenshot
    ▷ ControlShadow
    ▷ FileModel
    ▷ ThemeManager
    ▷ ThemeManager.ThemingEventHandler
```

في حين أن مجال الأسماء Eng27.UI فهو عصب المكتبة:

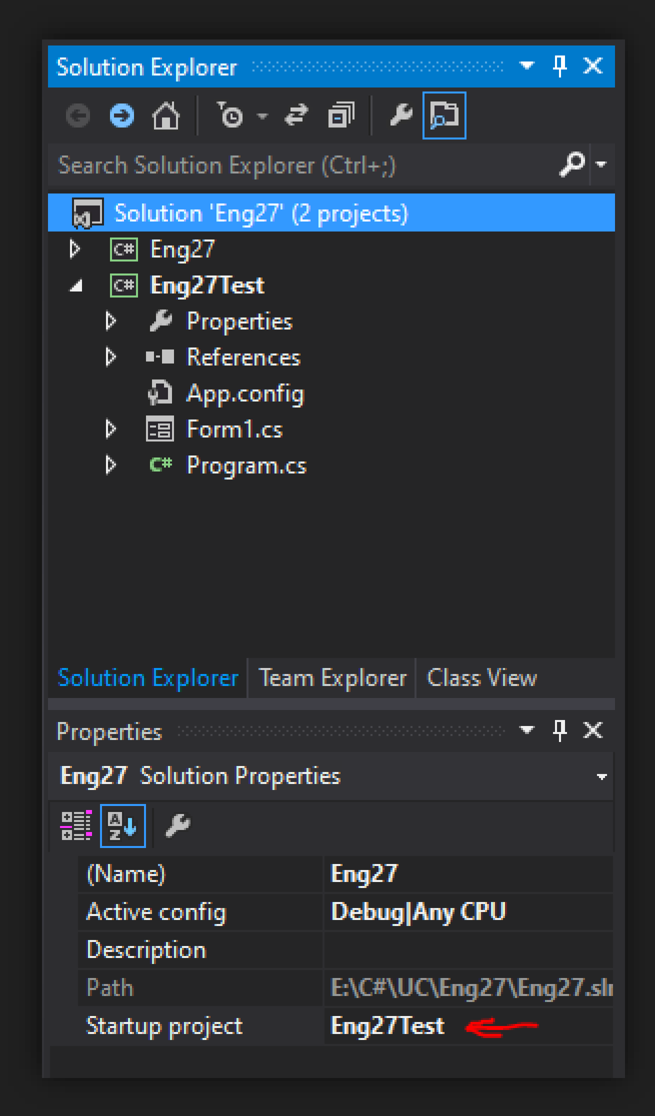
```
{ } Eng27.UI
    ButtonAnimated
    ButtonAnimated.ThemingEventHandler
    ButtonCircular
    ButtonCircular.ThemingEventHandler
    ButtonCircularAnimated
    ButtonCircularAnimated.ThemingEventHandler
    ButtonDoubleAnimated
    ButtonDoubleAnimated.ButtonDoubleAnimatedEventHandler
    ButtonDoubleAnimated.ThemingEventHandler
    ButtonFlat
    ButtonGradiant
    ButtonImage
    ButtonSquare
    ChartPie
    CheckBoxCircular
    ColorBoard
    DigitPainter
    Eng27Form
```

```
    FormShadow
    FormShadow.MARGINS
    ListBoxReArrange
    ListBoxTransfer
    PanelGradiant
    PanelRounded
    PanelShadow
    PictureBoxCircular
    ProgressBarCircular
    StepsProgress
    TextBoxLabeled
    TextBoxLine
    TextBoxRounded
    TextBoxRoundedButton
    TextBoxSingleLine
    TextBoxWatermark
    ToggleSwitch
    ToggleSwitchLabeled
    ToggleSwitchLabeled.ToggleChangedEventHandler
```

أما مجالا الأسماء Eng27.Enums وEng27.Interfaces فسيحويان معددات وواجهات المكتبة:



في البداية، أضف مشروعًا جديدًا من النوع WindowsForms إلى نفس المشروع، واجعله المشروع الافتراضي، كما في الشكل المجاور:



واجهات المكتبة

للواجهات دور في تمييز فئات المكتبة عن بعضها، وإعطاءها صفات معينة. أنشأنا فئتين في ما مضى: الواجهة IEng27Control والواجهة IEng27Component، هناك واجهات أخرى سنحتاجها، سنسردها معًا في هذه الفقرة.

الواجهة IEng27Animation

ستجبر هذه الواجهة الأدوات التي ترث منها على التحلي بأعضاء معينة تجعلها متحركة أو لها سمات حركية:



```
namespace Eng27.Interfaces
{
    /// <summary>
    /// Provides an animation to Eng27 controls.
    /// </summary>
    public interface IEng27Animation
    {
        /// <summary>
        /// Gets or sets the time, in melliseconds, before
        /// next step.
        /// </summary>
        int Interval { get; set; }
        /// <summary>
        /// Gets a value indecate that the control is animating.
        /// </summary>
        bool IsAnimating { get; }
        /// <summary>
        /// Gets or set a value indecates the steps of the control.
        /// </summary>
        int Steps { get; set; }
    }
}
```

الواجهة IEng27Border

ستمثل هذه الواجهة الأدوات التي تحوي حوافًا:



```
using System.Drawing;

namespace Eng27.Interfaces
{
    ///
    public interface IEng27Border
    {
        int BorderSize { get; set; }
        Color BorderColor { get; set; }
    }
}
```

الواجهة IEng27Form

ستميز نوافذ المكتبة، لاحظ أن الواجهة ترث من الواجهة الأولى:



```
namespace Eng27.Interfaces
{
    ///
    public interface IEng27Form: IEng27Control {
    }
}
```

الواجهة IEng27Radial

ستميز الأدوات ذات الشكل الدائري:



```
namespace Eng27.Interfaces
{
    ///
    public interface IEng27Radial : IEng27Control
    {
        int Radius { get; set; }
    }
}
```

النماذج Forms

النموذج Eng27Form



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System.ComponentModel;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class Eng27Form : Form, IEng27Form
    {
        #region Constructors
        ///
        public Eng27Form() {

        }
        #endregion


        // نفس خصائص الأداة
        // Eng27Control
        #Properties#


        #region Methods
        void ChangeStyle()
        {
            switch (style)
            {
                case Style_e.Dark:
                    this.BackColor = styledark.BackColor;
                    this.ForeColor = styledark.ForeColor;
                    this.Font = styledark.Font;
                    break;
```

```
                case Style_e.Light:
                    this.BackColor = stylelight.ForeColor;
                    this.ForeColor = stylelight.BackColor;
                    this.Font = stylelight.Font;
                    break;

                default:
                    break;
            }
        }
        #endregion


        #region Events
        protected override void OnCreateControl()
        {
            this.ChangeStyle();
            base.OnCreateControl();
        }

        // ضع أحداث الأداة
        // Eng27Control
        void StyleLight_StyleChanged(object sender, System.EventArgs e) ...
        void StyleDark_StyleChanged(object sender, System.EventArgs e) ...
        void thememanager_ThemeChanged(object sender, ThemingEventArgs e) ...
        #endregion
    }
}
```

هناك الكثير من الأحداث التي يمكنك إنشاؤها، مثل حدث OnControlCreation أو حدث OnControlDeath، وذلك اعتمادًا على التوابع البنّاءة والتوابع الهدّامة[1]، هذا إن لم تكن أدواتك تحوي هذه الأحداث. في الواقع، كل خاصية من خصائص فئتك تعطيك إمكانية إنشاء حدثين على الأقل: حدث عند طلب البيانات (الكلمة get)، وحدث عند تغيير البيانات (الكلمة set)، بالإضافة إلى أحداث عند بداية ضبط البيانات وأحداث عند نهايتها (قد يتم تنفيذ بعض الأكواد في قسم الكلمة set من الخاصية، عندها يمكنك تفجير الحدث OnPropertyChanging عند بداية تغيير البيانات، والحدث OnPropertyChanged عند نهايتها).

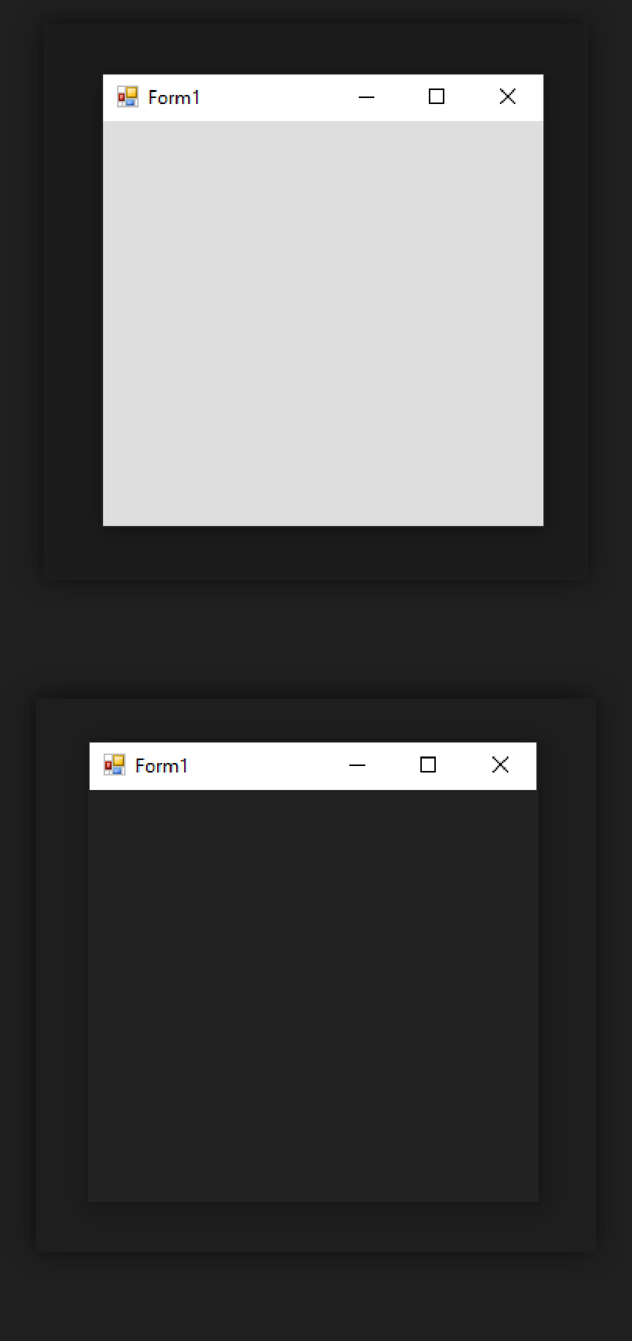
لاحظ أن هذه الفئة تقوم بتطبيق المظهر بشكل معاكس في النمط المضيء StyleLight، فهي تضبط لون الخط بلون الخلفية، والعكس بالعكس. يمكنك تطوير الأداة

[1] التابع البنّاء Contructor والتابع الهدّام Destructor هي توابع تستدعى عند إنشاء وعند فناء الفئة، انظر كتابنا "C# من البداية حتى الإتقان" ص. 190-191 للمزيد.

ThemeManager لتحوي خصائص خاصة بالنوافذ والأدوات الحاوية عوضًا عن الاعتماد على هذا الأسلوب، إلا أني سأعتمد عليه لاختصار الفئات.

الحدث OnControlCreation يساهم في ضبط النمط عند إنشاء الأداة.

إذا أعدت بناء المشروع، وانتقلت لمشروع النوافذ، وجعلت النافذة Form1 ترث من الفئة Eng27Form، وضبطت مظهر النافذة مرة على النمط Light ومرة على النمط Dark وشغلت المشروع:



ستحتاج للتصريح عن مجال الأسماء Eng27.UI باستخدام الكلمة using.

النموذج FormShadow



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Runtime.InteropServices;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class FormShadow : Form, IEng27Form
    {
        #region Local Variables
        // متغيرات لضبط الظل
        private bool m_aeroEnabled;
        private const int CS_DROPSHADOW = 0x00020000;
        private const int WM_NCPAINT = 0x0085;
        private const int WM_ACTIVATEAPP = 0x001C;

        // متغيرات لضبط سحب وإلقاء النافذة
        private const int WM_NCHITTEST = 0x84;
        private const int HTCLIENT = 0x1;
        private const int HTCAPTION = 0x2;
        #endregion


        #region DllImports
        [DllImport("Gdi32.dll", EntryPoint = "CreateRoundRectRgn")]
        private static extern IntPtr CreateRoundRectRgn
        (
            int nLeftRect, // x-coordinate of upper-left corner
            int nTopRect, // y-coordinate of upper-left corner
            int nRightRect, // x-coordinate of lower-right corner
            int nBottomRect, // y-coordinate of lower-right corner
            int nWidthEllipse, // height of ellipse
            int nHeightEllipse // width of ellipse
         );

        [DllImport("dwmapi.dll")]
        public static extern int DwmExtendFrameIntoClientArea
            (IntPtr hWnd, ref MARGINS pMarInset);

        [DllImport("dwmapi.dll")]
        public static extern int DwmSetWindowAttribute
            (IntPtr hwnd, int attr, ref int attrValue, int attrSize);

        [DllImport("dwmapi.dll")]
        public static extern int DwmIsCompositionEnabled
            (ref int pfEnabled);
        #endregion
```

```
        #region Constructors
        ///
        public FormShadow()
        {
            m_aeroEnabled = false;

            Screen screen = Screen.FromControl(this);
            int x = screen.WorkingArea.X - screen.Bounds.X;
            int y = screen.WorkingArea.Y - screen.Bounds.Y;
            this.MaximizedBounds = new Rectangle(x, y,
                screen.WorkingArea.Width, screen.WorkingArea.Height);
            this.MaximumSize = screen.WorkingArea.Size;

            this.FormBorderStyle = FormBorderStyle.None;
        }
        #endregion


        // نفس خصائص الأداة
        // Eng27Control
        #Properties#


        #region Structs
        // من أجل الظل
        public struct MARGINS
        {
            public int leftWidth;
            public int rightWidth;
            public int topHeight;
            public int bottomHeight;
        }
        #endregion


        #region Methods
        protected override CreateParams CreateParams
        {
            get
            {
                m_aeroEnabled = CheckAeroEnabled();

                CreateParams cp = base.CreateParams;
                if (!m_aeroEnabled)
                    cp.ClassStyle |= CS_DROPSHADOW;

                return cp;
            }
        }

        private bool CheckAeroEnabled() {
            if (Environment.OSVersion.Version.Major >= 6)
            {
                int enabled = 0;
                DwmIsCompositionEnabled(ref enabled);
                return (enabled == 1) ? true : false;
            }
            return false;
        }
```

```
        protected override void WndProc(ref Message m) {
            switch (m.Msg) {
                case WM_NCPAINT:
                    if (m_aeroEnabled)
                    {
                        var v = 2;
                        DwmSetWindowAttribute(this.Handle, 2, ref v, 4);
                        MARGINS margins = new MARGINS()
                        {
                            bottomHeight = 1,
                            leftWidth = 1,
                            rightWidth = 1,
                            topHeight = 1
                        };
                        DwmExtendFrameIntoClientArea(this.Handle, ref margins);

                    }
                    break;
                default:
                    break;
            }
            base.WndProc(ref m);

            if (m.Msg == WM_NCHITTEST && (int)m.Result == HTCLIENT)
                m.Result = (IntPtr)HTCAPTION;
        }

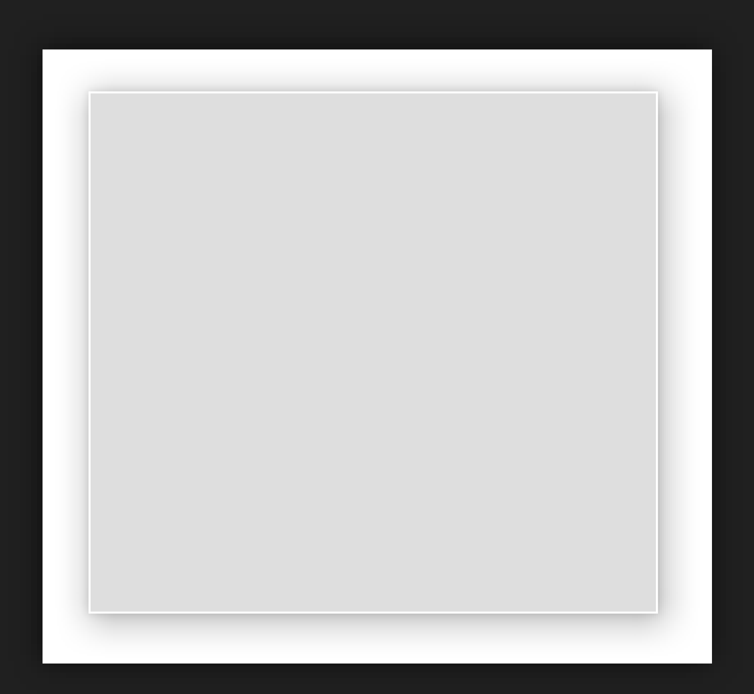
        void ChangeStyle() {
            switch (style)
            {
                case Style_e.Dark:
                    this.BackColor = styledark.BackColor;
                    this.ForeColor = styledark.ForeColor;
                    this.Font = styledark.Font;
                    break;

                case Style_e.Light:
                    this.BackColor = stylelight.ForeColor;
                    this.ForeColor = stylelight.BackColor;
                    this.Font = stylelight.Font;
                    break;

                default:
                    break;
            }
        }
        #endregion


        #region Events
        //  ضع أحداث الأداة
        // Eng27Control
        void StyleLight_StyleChanged(object sender, System.EventArgs e) ...
        void StyleDark_StyleChanged(object sender, System.EventArgs e) ...
        void thememanager_ThemeChanged(object sender, ThemingEventArgs e) ...
        #endregion
    }
}
```

في مشروع النوافذ، غيّر مصدر الوراثة واجعل النافذة Form1 ترث من ShadowForm، وشغل المشروع:



ما يميز هذه النافذة – فضلًا عن تأثير الظل الجميل فيها – هو إمكانية نقلها مع أنها لا تحوي شريط عنوان كالنوافذ القياسية.

أدوات قياسية مطورة

جميع أدوات هذه الفقرة مألوفة ومعروفة، وما سنقوم به هو تعديل بعض خصائصها وأعضاءها.

ButtonFlat

هذه الأداة ليست إلا زرًّا عاديًّا، مسطحًّا بطبيعته، وله خصائص تضبط مظهره.

```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace Eng27.UI
{
```

```
/// <summary>
/// Represents an Eng27 Flat button control.
/// </summary>
[DefaultEvent("Click")]
[DefaultProperty("Text")]
[DesignerCategory("Eng27 Control")]
[ToolboxBitmap(typeof(Button))]
public class ButtonFlat : Eng27ButtonBase, IEng27Border
{
    #region Constructors
    /// <summary>
    /// Initializes a new instance of the Eng27.UI.ButtonFlat class.
    /// </summary>
    public ButtonFlat()
    {
        this.Width = 100;
        this.Style = Style_e.Light;
        this.StyleDark.StyleChanged += StyleDark_StyleChanged;
        this.StyleLight.StyleChanged += StyleLight_StyleChanged;
        bordercolor = this.ForeColor;
        /*
          ملاحظة
         */
    }
    #endregion


    #region Properties
    private ButtonBorderStyle buttonborderstyle = ButtonBorderStyle.Solid;
    ///
    public ButtonBorderStyle ButtonBorderStyle
    {
        get { return buttonborderstyle; }
        set { buttonborderstyle = value; this.Invalidate(); }
    }

    private int bordersize = 1;
    ///
    public int BorderSize
    {
        get { return bordersize; }
        set { bordersize = value; this.Invalidate(); }
    }

    private Color bordercolor;
    ///
    public Color BorderColor
    {
        get { return bordercolor; }
        set { bordercolor = value; this.Invalidate(); }
    }

    // نفس خصائص المظهر من الأداة
    // Eng27Control

    public ThemeManager ThemeManager ...
    public StyleDark StyleDark ...
    public StyleLight StyleLight ...
    public Style_e Style ...
```

```
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public FlatStyle FlatStyle { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public FlatButtonAppearance FlatAppearance { get; set; }
        #endregion


        #region Methods
        void ChangeStyle()
        {
            switch (style)
            {
                case Style_e.Dark:
                    this.BackColor = styledark.BackColor;
                    this.ForeColor = styledark.ForeColor;
                    this.Font = styledark.Font;
                    break;

                case Style_e.Light:
                    this.BackColor = stylelight.BackColor;
                    this.ForeColor = stylelight.ForeColor;
                    this.Font = stylelight.Font;
                    break;

                default:
                    break;
            }

            if (thememanager != null)
            {
                if (thememanager.UserBorderSettings)
                    this.BorderColor = thememanager.BorderColor;
                else
                    this.BorderColor = this.ForeColor;
                this.BorderSize = thememanager.BorderSize;
            }

            else
                this.BorderColor = this.ForeColor;
        }
        #endregion


        #region Events
        protected override void OnPaint (PaintEventArgs pevent)
        {
            base.OnPaint(pevent);
            ControlPaint.DrawBorder(
                pevent.Graphics, ClientRectangle,
                       bordercolor, bordersize, buttonborderstyle,
                       bordercolor, bordersize, buttonborderstyle,
                       bordercolor, bordersize, buttonborderstyle,
                       bordercolor, bordersize, buttonborderstyle);
        }

        void StyleLight_StyleChanged(object sender, System.EventArgs e) ...
        void StyleDark_StyleChanged(object sender, System.EventArgs e) ...
```

```
        private void thememanager_ThemeChanged(object sender, ThemingEventArgs e)
        {
            switch (e.Theming)
            {
                case ThemingArgs_e.StyleChanged:
                    if (this.Theming == Theming_e.AllowBorderOnly ||
                        this.Theming == Theming_e.Disallow)
                        break;

                    styledark = thememanager.StyleDark;
                    stylelight = thememanager.StyleLight;
                    style = thememanager.Style;
                    this.ChangeStyle();
                    break;

                case ThemingArgs_e.BorderSizeChanged:
                    if (this.Theming == Theming_e.AllowStyleOnly ||
                        this.Theming == Theming_e.Disallow)
                        break;
                    bordersize = thememanager.BorderSize;
                    break;

                case ThemingArgs_e.BorderColorChanged:
                    if (this.Theming == Theming_e.AllowStyleOnly ||
                        this.Theming == Theming_e.Disallow)
                        break;

                    bordercolor = thememanager.BorderColor;
                    break;

                default:
                    break;
            }
        }
        #endregion
    }
}
```

ملاحظة:

- عند الوصول للمتغيرات دون الخصائص فإن أحداث الفئة لن يتم تفجيرها، لذلك فالأفضل الوصول للخصائص وليس المتغيرات، إلا إذا كانت الخاصية لا تقوم بوظائف أخرى (استدعاء طرق أو التحقق من شروط أو تفجير أحداث) عندها لا فرق بين الخصائص والمتغيرات.

فمثلًا، لاحظ أن الخصائص BorderSize وBorderColor تقوم باستدعاء الطريقة Invalidate لإجبار الأداة على رسم نفسها (تفجير الحدث OnPaint)، لذلك فإذا أسندت قيمة ما

للمتغيرات المتعلقة بهذه الخصائص فلن يتم استدعاء هذه الطريقة ولن يتم إعادة رسم الأداة، على عكس ما لو تم إسناد القيمة للخصائص.

في الواقع، فإن التعامل مع المتغيرات دون الخصائص ليس آمنًا، فتجنب التعامل معها قدر الإمكان.

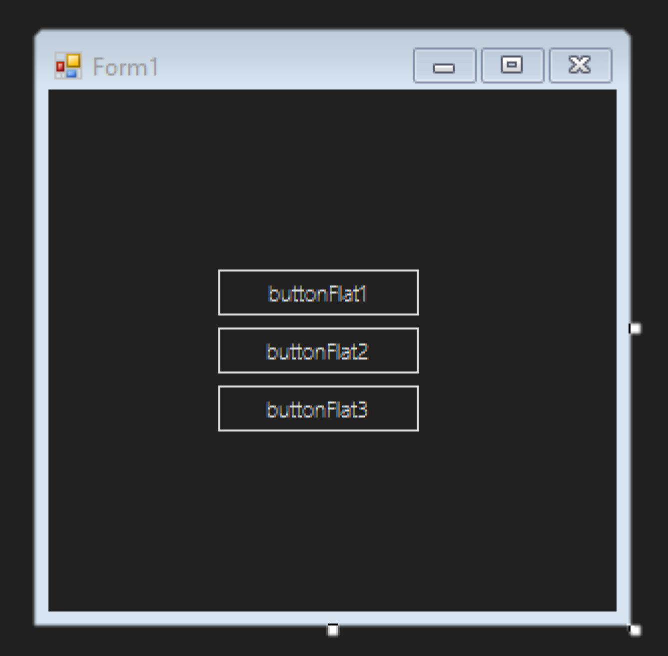
على اعتبار الأداة أداة حواف – لاتصافها بأعضاء الواجهة IEng27Border – فقد زودنا أداة مدير السمات ThemeManager بخاصية تعطي المبرمج إعدادات الحواف، ولكن أولًا عليك التأكد من أن الكائن المُسنَد للخاصية ThemeManager يمثل شيئًا (ليس null)؛ إذ إن الإجراء ChangeStyle يُستدعى سواء كان هناك كائن أم لا.

كما أن هذه الأداة يجب ألا تكون إلا مسطحة Flat، لذلك فأخفينا خصائص المظهر المسطح FlatStyle وFlatAppearance.

أعد بناء المشروع، وانتقل لمشروع النوافذ الذي أنشأته، ولاحظ أدواتنا:

Eng27 Components
Pointer
ThemeManager
ButtonFlat

لاحظ أن الأدوات لا تملك أيقونات لأنها في نفس المشروع. أنشئ ثلاثة أزرار، ومدير سمات، كما يلي:



القيمة	الخاصية	الأداة
themeManager1	ThemeManager	Form1
Red	BorderColor	themeManager1
True	UserBorderSettings	
themeManager1	ThemeManager	buttonFlat1
AllowFull	Theming	
themeManager1	ThemeManager	buttonFlat2
AllowBorderOnly	Theming	
themeManager1	ThemeManager	buttonFlat3
Disallow	Theming	



```
using Eng27.Enums;
using Eng27.UI;
using System;
using System.Windows.Forms;

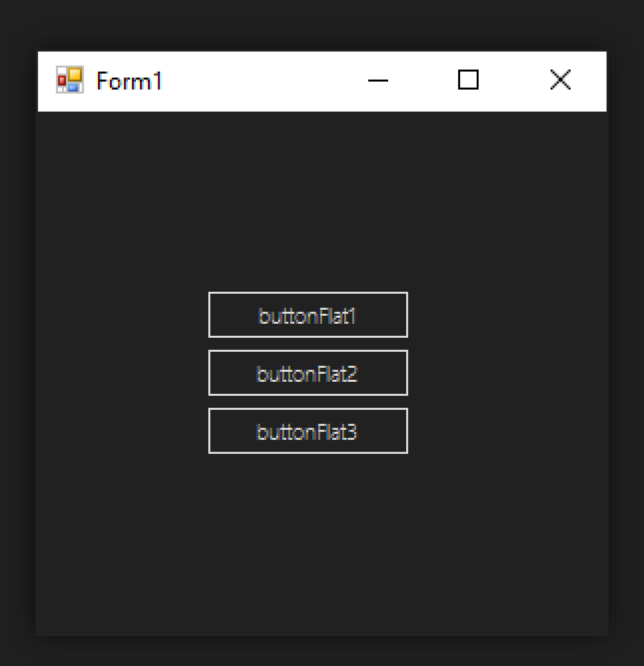
namespace Eng27Test
{
    public partial class Form1 : Eng27Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void buttonFlat1_Click(object sender, EventArgs e)
        {
            themeManager1.Style = Style_e.Dark;
        }

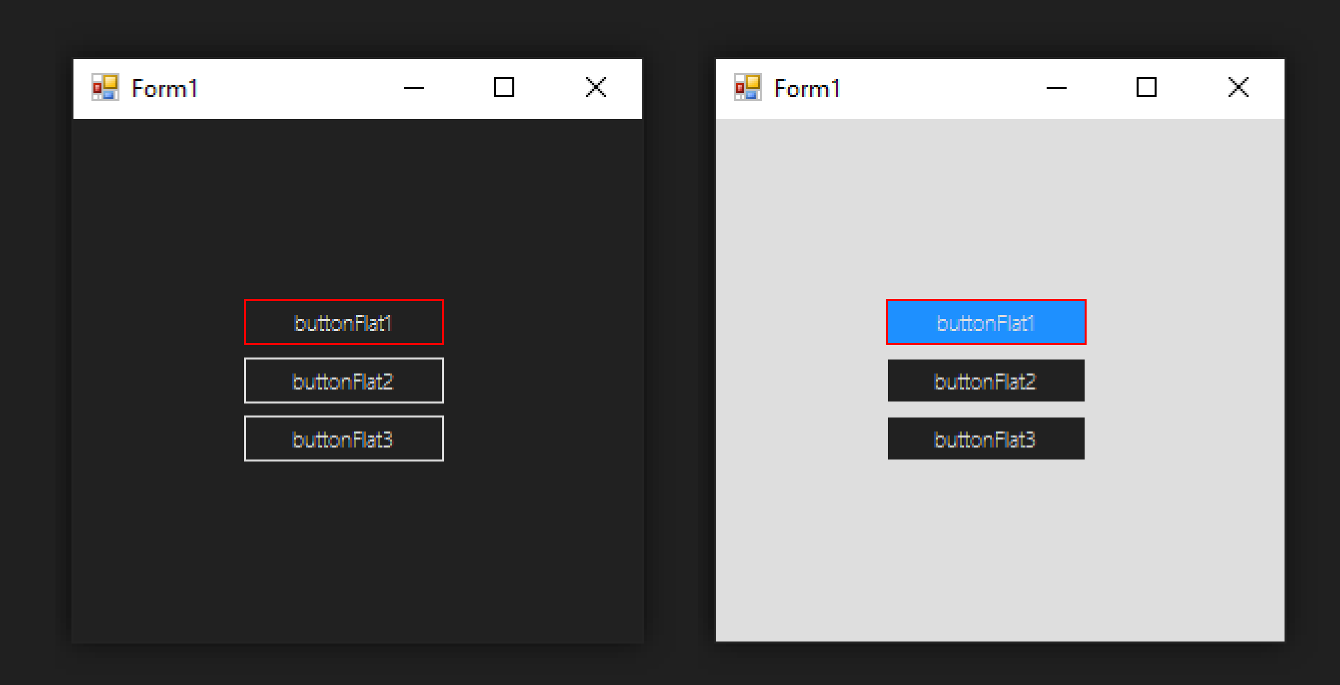
        private void buttonFlat2_Click(object sender, EventArgs e)
        {
            themeManager1.Style = Style_e.Light;
        }
    }
}
```

شغّل المشروع، وانقر على الزر الأول ثم الثاني، ولاحظ:

عند تشغيل البرنامج، وعلى اعتبار أن القيمة الافتراضية لمدير السمات هي Dark، فستحصل على:



عند النقر على الأزرار:



ButtonSquare

لا تختلف هذه الأداة عن الأداة Button القياسية شيئًا، كما أنها لا تدعم المظهر – لغاية في نفسي – وبطبيعة الحال لا تدعم مدير السمات. هي فقط زر قياسي مربع دائمًا.

يمكن الوصول للنتيجة من خلال الوراثة من الفئة Button، ولكن لجعل الأداة تتسم بالواجهة IEng27Control سنجعلها ترث من الفئة Eng27ButtonBase.



```
using Eng27.Components;
using Eng27.Enums;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class ButtonSquare : Eng27ButtonBase
    {
        #region Constructors
        ///
        public ButtonSquare()
        {
            this.Height = 100;
            this.FlatStyle = FlatStyle.Standard;
            this.BackColor = SystemColors.Control;
            this.ForeColor = Color.Black;
        }
        #endregion


        #region Properties
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public ThemeManager ThemeManager { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Theming_e Theming { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleDark StyleDark { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleLight StyleLight { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Style_e Style { get; set; }
        #endregion
```

```
        #region Methods
        protected override void SetBoundsCore
            (int x, int y, int width, int height, BoundsSpecified specified)
        {
            width = height;
            base.SetBoundsCore(x, y, width, height, specified);
        }
        #endregion
    }
}
```



يمكنك إنشاء فئة باسم Eng27NotThemableControl تضع فيها خصائص المظهر بعد توصيفها بالمواصفات التي تلغيها.

إذا اعتمدت على الفئة Button عوضًا عن Eng27ButtonBase فإنك لن تحتاج إلا كود ضبط الارتفاع ضمن التابع البناء Constructor و الإجراء SetBoundsCode.

TextBoxSingleLine

صندوق النص هذا سطري دائمًا، وهذا يعني أن المبرمج لا يمكنه ضبطه على أنه متعدد الأسطر (من خلال الخاصية MultiLine)؛ وهذا يتطلب إخفاء الخاصية من نافذة الخصائص Property Windwos ولائحة المهام Task Pane.



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    [Designer(@"System.Windows.Forms.Design.ControlDesigner, System.Design," +
               " Version=2.0.0.0," +
               " Culture=neutral," +
               " PublicKeyToken=b03f5f7f11d50a3a")]
    public class TextBoxSingleLine:TextBox, IEng27Control {
        #region Constructors
        ///
        public TextBoxSingleLine()
        {
            styledark.BackColor = Color.White;
            styledark.ForeColor = Color.Black;
```

```
            stylelight.BackColor = Color.White;
            stylelight.ForeColor = Color.Black;

            this.StyleDark.StyleChanged += StyleDark_StyleChanged;
            this.StyleLight.StyleChanged += StyleLight_StyleChanged;
        }
        #endregion


        #region Properties
        [Browsable(false)]
        [EditorBrowsable(EditorBrowsableState.Never)]
        public override bool Multiline
        {
            get { return base.Multiline; }
            set { base.Multiline = value; }
        }

        // نفس خصائص المظهر من الأداة
        // Eng27Control

        public ThemeManager ThemeManager ...
        public StyleDark StyleDark ...
        public StyleLight StyleLight ...
        public Style_e Style ...
        public Theming_e Theming ...
        #endregion


        #region Methods
        void ChangeStyle() ...
        #endregion

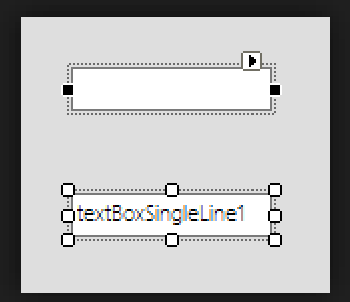

        #Events
        void StyleLight_StyleChanged ...
        void StyleDark_StyleChanged ...
        private void thememanager_ThemeChanged ...
        #endregion
    }
}
```

على اعتبار أن الفئة ButtonFlat تعود إلى فئة أم فيها توصيف أعضاء المظهر (الخصائص StyleDark وStyleLight) فلسنا بحاجة لتوصيف هذه الأعضاء في الفئة الابن، بينما لا نجد ذلك في الفئة TextBoxSingleLine، لذلك فعلينا توصيف الأعضاء، وإلا لما تمكّنا من الوصول لجزئيات الخصائص وتفاصيلها.

ونأتي لمربط الفرس، فالمميز في هذه الأداة أنه لا يمكن للمستخدم جعلها متعددة الأسطر MultiLine من خلال المصمم، خصوصًا من خلال الأوامر المميزة Smart Tag. في أداة صندوق النص TextBox التقليدية إذا كانت الأداة متعددة الأسطر (أي أن

(MultiLine = true) فإنه لا يمكن استخدام صندوق النص كصندوق نص فيه كلمة مرور، ولن تستطيع الوصول لهذه الإمكانية إلا إذا أزلت الخاصية MultiLine، وهذا منطقي.

لاحظ الفرق بين الأداة TextBox والأداة TextBoxSingleLine في طور التصميم:



أدوات مدمجة

عندما تشترك أكثر من أداة في تكوين أداة جديدة نحصل على ما يسمى بالأدوات المدمجة أو الهجينة. والأداة المستخدمة لتكوين الأدوات الهجينة هي UserControl. الأداة UserControl تعود بالأساس إلى الفئة Control، لكن بينهما أكثر من فئة (الأداة UserControl ترث من فئة، ترث من فئة، وهكذا حتى الوصول للفئة Control).

وهذا – كونها تعود للفئة Control – يعطينا نتيجتين، الأولى: أن الأداة UserControl ذات طبيعة رسومية؛ إذ إن الفئة الأولى هي ما تعطي ما يرث منها طبيعته الرسومية، وبعضَ الأعضاء القياسية الأخرى (مجموعة من الخصائص والطرق والأحداث المختلفة التي تجدها في أغلب الأدوات ذات الطبيعة الرسومية)، الثانية: أن الفئات التي تفصل هذه الأداة عن الفئة الأولى هي الفئات التي تعطيها قابليتها لاحتواء الأدوات، بالإضافة إلى أعضاء أخرى غير موجودة ضمن أعضاء الفئة الأم.

Eng27UserControl

على اعتبار أن أدواتنا فيها بعض الأعضاء المشتركة (والتي أضفناها من خلال الواجهات، مثل المظهر Style)، فإن على الأداة الخاصة التي سننشئ منها أدواتنا الهجينة أن تحوي

هذه الأعضاء أيضًا، لذلك فعلينا إنشاء أداة جديدة معدلة عن الأداة العادية نعتمد عليها في تصميم أدواتنا الهجينة.



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System.ComponentModel;
using System.Windows.Forms;

namespace Eng27
{
    ///
    public class Eng27UserControl: UserControl, IEng27Control
    {
        #region Constructors
        ///
        public Eng27UserControl()
        {
            this.AutoScaleMode = AutoScaleMode.None;
        }
        #endregion


        #region Properties
        public ThemeManager ThemeManager ...
        public Style_e Style ...
        public StyleDark StyleDark ...
        public StyleLight StyleLight ...
        public Theming_e Theming ...
        #endregion


        #region Methods
        public virtual void ChangeStyle() ...
        #endregion


        #region Events
        void StyleLight_StyleChanged(object sender, System.EventArgs e) ...
        void StyleDark_StyleChanged(object sender, System.EventArgs e) ...
        void thememanager_ThemeChanged(object sender, ThemingEventArgs e) ...
        #endregion
    }
}
```

TextBoxLabeled



```
using Eng27.Enums;
using System.ComponentModel;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class TextBoxLabeled : Eng27UserControl
    {
        #region Local Variables
        TextBox textbox;
        Label label;
        #endregion


        #region Constructors
        ///
        public TextBoxLabeled()
        {
            this.Height = 30;

            textbox = new TextBox() { Dock = DockStyle.Bottom };
            label = new Label() { Dock = DockStyle.Top, Text = "label" };

            Controls.AddRange(new Control[] { textbox, label });

            this.Height = textbox.Height + label.Height;
        }
        #endregion


        #region Properties
        private string labeltext;
        ///
        public string LabelText
        {
            get { return labeltext; }
            set { labeltext = value; label.Text = value; }
        }

        ///
        [EditorBrowsable(EditorBrowsableState.Always)]
        [Browsable(true)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
        [Bindable(true)]
        public override string Text
        {
            get { return textbox.Text; }
            set { textbox.Text = value; }
        }
        #endregion

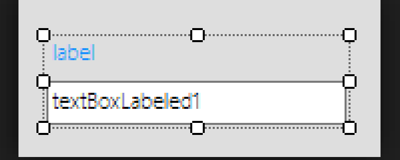

        #region Methods
        public override void ChangeStyle()
        {
```

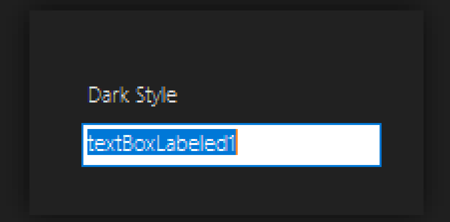
```
            switch (this.Style)
            {
                case Style_e.Dark:
                    this.BackColor = this.StyleDark.BackColor;
                    this.ForeColor = this.StyleDark.ForeColor;
                    this.Font = this.StyleDark.Font;
                    break;

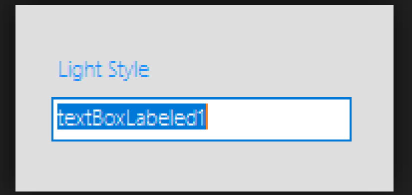
                case Style_e.Light:
                    this.BackColor = this.StyleLight.ForeColor;
                    this.ForeColor = this.StyleLight.BackColor;
                    this.Font = this.StyleLight.Font;
                    break;

                default:
                    break;
            }
        }
        #endregion
    }
}
```

على اعتبار أن هذه الأداة هي حواية لغيرها من الأدوات، فإن لون خلفيتها يجب أن يكون مخالفًا للون خلفية المظهر في الوضع المضيء Light.







TextBoxLine



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class TextBoxLine:Eng27UserControl, IEng27Border
    {
        #region Local Variables
        TextBox textbox;
        #endregion


        #region Constructors
        ///
        public TextBoxLine() {
            textbox = new TextBox() { Dock = DockStyle.Top };

            Controls.Add(textbox);

            textbox.SizeChanged += textbox_SizeChanged;
            textbox.GotFocus += textbox_GotFocus;
            textbox.Leave += textbox_Leave;
        }
        #endregion


        #region Properties
        private int bordersize = 2;
        ///
        public int BorderSize
        {
            get { return bordersize; }
            set { bordersize = value; }
        }

        private Color bordercolor = Color.Blue;
        ///
        public Color BorderColor
        {
            get { return bordercolor; }
            set { bordercolor = value; }
        }

        private Color entercolor = Color.Red;
        ///
        public Color EnterColor
        {
            get { return entercolor; }
            set { entercolor = value; }
        }
```

```
        private Color leavecolor = Color.Blue;
        ///
        public Color LeaveColor
        {
            get { return leavecolor; }
            set { leavecolor = value; }
        }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public ThemeManager ThemeManager { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Theming_e Theming { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleDark StyleDark { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleLight StyleLight { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Style_e Style { get; set; }
        #endregion


        #region Events
        void textbox_GotFocus(object sender, EventArgs e)
        {
            bordercolor = entercolor; this.Invalidate();
        }

        void textbox_Leave(object sender, EventArgs e)
        {
            bordercolor = leavecolor; this.Invalidate();
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            base.OnPaint(e);
            Graphics g = e.Graphics;
            Pen pen = new Pen(bordercolor, bordersize);
            g.DrawLine(
                pen,
                new Point(0, textbox.Height + bordersize / 2),
                new Point(this.Width, textbox.Height + bordersize / 2)
                );
        }

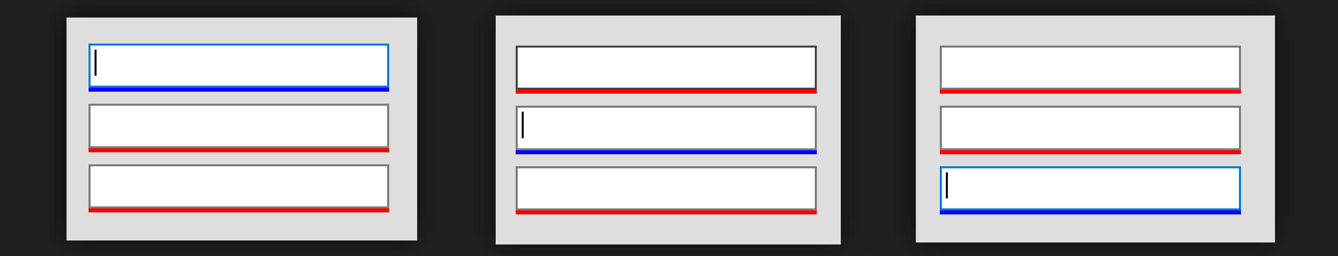
        protected override void OnSizeChanged(EventArgs e)
        {
            base.OnSizeChanged(e);
            if (textbox != null)
                this.Height = textbox.Height + bordersize;
        }
```

```
        void textbox_SizeChanged(object sender, EventArgs e)
        {
            this.OnSizeChanged(e);
        }
        #endregion
        /*
          ملاحظة
         */
    }
}
```

ملاحظة:

- زود هذه الأداة بأحداث تراقب متى يمر مؤشر الفأرة فوق الأداة ومتى ينتقل إليها التركيز وغيرها، وغير لون الخط بناءً على ذلك.

أضف ثلاثة صناديق نصوص من هذا النوع، وتنقّل بينها:



TextBoxWatermark

العلامة المائية Water Mark هي عبارة نصية تكتب بلون خفيف أو شفاف Transparent، تجدها على الصور وصفحات الكتب والمجلات وغيرها، وفي البرامج على صناديق الإدخال، فهي تعطي المستخدم معلومات إضافية عن الأداة التي يستخدمها.

```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace Eng27.UI
{
```

```
    ///
    public class TextBoxWatermark : TextBox, IEng27Control
    {
        #region Local Variables
        Label label;
        #endregion


        #region Constructors
        ///
        public TextBoxWatermark() {
            watermark = "WaterMark";
            watermarkmargin = 3;
            label = new Label
            {
                BackColor = Color.Transparent,
                ForeColor = Color.Gray,
                Text = watermark,
                Left = watermarkmargin
            };

            Controls.Add(label);

            this.TextChanged += WatermarkTextBox_TextChanged;
            label.Click += label_Click;
        }
        #endregion


        #region Properties
        private int watermarkmargin;
        ///
        public int WaterMarkMargin
        {
            get { return watermarkmargin; }
            set { watermarkmargin = value; label.Left = value; }
        }

        private string watermark;
        ///
        public string WaterMark
        {
            get { return watermark; }
            set { watermark = value; label.Text = value; }
        }

        private Color watermark_color = Color.Gray;
        ///
        public Color WatermarkColor
        {
            get { return watermark_color; }
            set { watermark_color = value; label.ForeColor = value; }
        }

        public ThemeManager ThemeManager ...
        public Style_e Style ...
        public StyleDark StyleDark ...
        public StyleLight StyleLight ...
        public Theming_e Theming ...
        #endregion
```

```
        #region Methods
        void ChangeStyle()
        {
            switch (style)
            {
                case Style_e.Dark:
                    this.BackColor = styledark.BackColor;
                    this.ForeColor = styledark.ForeColor;
                    this.Font = styledark.Font;
                    this.WatermarkColor = Color.Gray;
                    break;

                case Style_e.Light:
                    this.BackColor = stylelight.BackColor;
                    this.ForeColor = stylelight.ForeColor;
                    this.Font = stylelight.Font;
                    this.WatermarkColor = Color.FromArgb(10,10,10);
                    break;

                default:
                    break;
            }
        }
        #endregion


        #region Events
        private void WatermarkTextBox_TextChanged(object sender, EventArgs e)
        {
            if (Text.Length > 0)
                label.Visible = false;
            else
                label.Visible = true;
        }

        void StyleLight_StyleChanged(object sender, System.EventArgs e) ...
        void StyleDark_StyleChanged(object sender, System.EventArgs e) ...
        void thememanager_ThemeChanged(object sender, ThemingEventArgs e) ...

        void label_Click(object sender, EventArgs e) {
            this.Focus();
        }
        #endregion
    }
}
```

لاحظ الأداة إذا لم يكن فيها محتوى نصي وإذا كان فيها:

يمكنك تطوير الأداة بجعلها تدعم المحاذاة، بحيث تكون العلامة المائية بنفس جهة النص.

هناك أسلوب آخر لتصميم هذه الأداة بالاعتماد على أحداث صندوق النص نفسه، وهو ما ناقشناه في مثال في الفصل الثاني، وذلك كما يلي:

- إذا لم تكن هناك قيمة نصية ضمن الأداة ولا يوجد تركيز عليها؛ تكون القيمة النصية لصندوق النص هي العلامة المائية ولون الخط هو لون العلامة المائية.
- إذا انتقل التركيز إلى الأداة، وكانت القيمة النصية هي العلامة المائية ولون الخط هو لون العلامة المائية؛ تُمسح القيمة النصية ويصبح لون الخط هو لون الخاصية ForeColor.
- إذا تم إزالة التركيز من الأداة، وكانت القيمة النصية هي قيمة فارغة؛ تصبح القيمة النصية هي العلامة المائية ولون النص هو لونها.

لكن، الاعتماد على أداة Label أفضل، إذ إنه – عند اعتماد الأسلوب الأول – عليك تطوير طريقة تقوم بجعل القيمة النصية المنسوخة من الأداة – عند النسخ – هي قيمة نصية فارغة إذا كانت القيمة النصية هي العلامة المائية. كما يمكنك أيضًا إنشاء متغيرات منطقية Boolean تقوم بتحديد حالة القيمة النصية لصندوق النص، هل هي قيمة نصية عادية أم علامة مائية.

وعلى كل حال، أي أسلوب ستعتمده طالما يوصلك للنتيجة، فلا بأس في ذلك، فكله عند العرب سوفتوير!

ListBoxReArrange

قد تكون لديك أداة صندوق لائحة ListBox، وترغب بتغيير موقع بعض عناصرها، لا تدعم الفئة ListBox طريقة لتغيير ترتيب عناصرها، لذلك عليك تطوير طريقة بنفسك.

يمكننا الوصول لهذه الغاية إما بتأليف أداة ترث من الفئة ListBox، ثم إنشاء الطرق اللازمة لذلك، أو بإنشاء أداة تأخذ كائنا من النوع ListBox، وتضع فيها الطرق المطلوبة. وهنا سنعتمد الأسلوب الأخير.

أما عن آلية نقل العناصر، فهي ليست أكثر من تبديل قيمة العناصر مع بعضها.



```
using Eng27.Enums;
using System;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class ListBoxReArrange: Eng27UserControl
    {
        #region Local Variables
        ButtonFlat button1, button2;
        ListBox listbox;
        #endregion


        #region Constructors
        ///
        public ListBoxReArrange()
        {
            this.Width = 50;
            this.Height = 100;

            button1 = new ButtonFlat
            {
                Dock = DockStyle.Top,
                Height = 50,
                ButtonBorderStyle = buttonborderstyle,
                Text = "UP"
            };

            button2 = new ButtonFlat
            {
                Dock = DockStyle.Bottom,
                Height = 50,
                ButtonBorderStyle = buttonborderstyle,
                Text = "DOWN"
            };

            this.Controls.Add(button1);
            this.Controls.Add(button2);

            button1.MouseDown += button1_MouseDown;
            button2.MouseDown += button2_MouseDown;
        }
        #endregion


        #region Properties
        private bool autobuttonsheight = false;
        ///
        public bool AutoButtonsHeight
        {
            get { return autobuttonsheight; }
            set
            {
                autobuttonsheight = value;
                if (value) this.ButtonHeight = this.Height / 2;
            }
        }
```

```
        private int buttonHeight = 50;
        ///
        public int ButtonHeight
        {
            get { return buttonHeight; }
            set { buttonHeight = value; button1.Height = button2.Height = value; }
        }

        private ButtonBorderStyle buttonborderstyle = ButtonBorderStyle.Solid;
        ///
        public ButtonBorderStyle ButtonBorderStyle
        {
            get { return buttonborderstyle; }
            set
            {
                buttonborderstyle = value;
                button1.ButtonBorderStyle = value;
                button2.ButtonBorderStyle = value; }
        }

        private bool movetofirstandlast = true;
        ///
        public bool MoveToFirstAndLast
        {
            get { return movetofirstandlast; }
            set { movetofirstandlast = value; }
        }

        ///
        public ListBox ListBox
        {
            get { return listbox; }
            set { listbox = value; }
        }

        public ThemeManager ThemeManager ...
        public Style_e Style ...
        public StyleDark StyleDark ...
        public StyleLight StyleLight ...
        public Theming_e Theming ...
        #endregion


        #region Methods
        ///
        public void MoveItem
            (int index, ListBoxMoveItem_e movingstate, MouseButtons b)
        {
            if (ListBoxesNotNull())
            {
                if (movingstate == ListBoxMoveItem_e.Forward)
                {
                    if (b == MouseButtons.Left)
                    {
                        if (index >= 0)
                        {
                            string temp;
                            if (index + 1 < listbox.Items.Count)
                            {
                                temp = listbox.Items[index].ToString();
```

```
                    listbox.Items[index] = listbox.Items[index + 1];
                    listbox.Items[index + 1] = temp;
                    listbox.SelectedIndex += 1;
                }
            }
        }

        else if (b == MouseButtons.Right)
        {
            if (movetofirstandlast)
            {
                if (index >= 0)
                {
                    string temp;
                    for (int i=index; i<listbox.Items.Count - 1; i++)
                    {
                        temp = listbox.Items[i].ToString();
                        listbox.Items[i] = listbox.Items[i + 1];
                        listbox.Items[i + 1] = temp;
                        listbox.SelectedIndex += 1;
                    }
                }
            }
        }
    }

    else if (movingstate == ListBoxMoveItem_e.Backward)
    {
        if (b == MouseButtons.Left)
        {
            if (index >= 0)
            {
                string temp;
                if (index > 0)
                {
                    temp = listbox.Items[index].ToString();
                    listbox.Items[index] = listbox.Items[index - 1];
                    listbox.Items[index - 1] = temp;
                    listbox.SelectedIndex -= 1;
                }
            }
        }

        else if (b == MouseButtons.Right)
        {
            if (movetofirstandlast)
            {
                if (index >= 0)
                {
                    string temp;
                    for (int i = index; i > 0; i--)
                    {
                        temp = listbox.Items[i].ToString();
                        listbox.Items[i] = listbox.Items[i - 1];
                        listbox.Items[i - 1] = temp;
                        listbox.SelectedIndex -= 1;
                    }
                }
            }
        }
```

```
                }
            }
        }

        protected override void SetBoundsCore
            (int x, int y, int width, int height, BoundsSpecified specified)
        {
            if (autobuttonsheight)
            {
                this.ButtonHeight = height / 2;
            }
            else
            {
                if (height < 2 * buttonHeight)
                    height = 2 * buttonHeight;
            }
            base.SetBoundsCore(x, y, width, height, specified);
        }

        bool ListBoxesNotNull()
        {
            if (listbox == null)
            {
                return false;
            }
            else
                return true;
        }
        #endregion


        #region Events
        private void button2_MouseDown(object sender, MouseEventArgs e)
        {
            this.MoveItem(listbox.SelectedIndex,
                ListBoxMoveItem_e.Forward,
                e.Button);
        }

        void button1_MouseDown(object sender, MouseEventArgs e)
        {
            this.MoveItem(listbox.SelectedIndex,
                ListBoxMoveItem_e.Backward,
                e.Button);
        }

        protected override void OnSizeChanged(EventArgs e) {
            if (this.Height < 2 * buttonHeight)
                this.Height = 2 * buttonHeight;
            this.Invalidate();
        }

        void StyleLight_StyleChanged(object sender, System.EventArgs e) ...
        void StyleDark_StyleChanged(object sender, System.EventArgs e) ...
        void thememanager_ThemeChanged(object sender, ThemingEventArgs e) ...
        #endregion
    }
}
```

يمكنك تطوير الأداة بتمكين المبرمج من ضبط القيمة النصية الظاهرة على الأزرار، أو بجعلها صورًا، وغيرها. كما يمكنك تضمين طرق تسمح للمستخدم بمسح العناصر.

حتى لا ننشئ طريقتين تقومان بنفس المهمة (نقل العنصر، واحدة للأمام وواحدة للخلف)، فقد اعتمدنا على معدد يحدد اتجاه النقل:

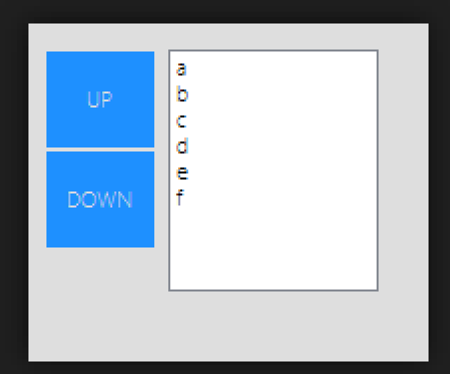
```
namespace Eng27.Enums
{
    ///
    public enum ListBoxMoveItem_e
    {
        Backward,
        Forward
    }
}
```

الطريقة MoveItem لا تدعم النقل بالاتجاهين فحسب، وإنما تقبل أيضًا النقل إلى بداية أو نهاية القائمة عوضًا عن النقل خطوة خطوة، وذلك من خلال الزر الذي سيتم نقره، فإذا تم النقر بالزر الأيسر تم النقل خطوة واحدة، أما إذا تم النقر بالزر الأيمن فيتم النقل إلى بداية أو نهاية القائمة. يمكنك تطوير الأداة لتسمح للمبرمج بتحديد الأزرار التي ستسبب النقل.

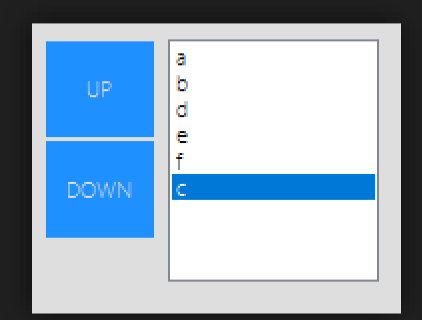


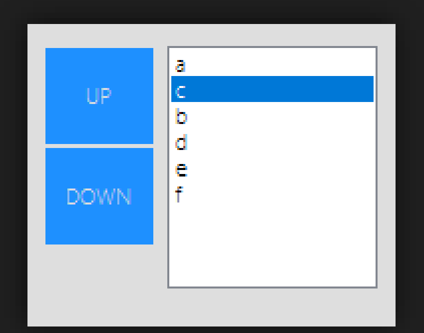
عند وجود شروط على المعددات، استخدم البنية switch وليس البنية if، مع أنك ستحصل على نفس النتيجة بالبنيتين.
يفضل استخدام بنية switch مع المجالات المحدودة (مثل المعددات، لها قيم محددة وليست لا نهائية)، أما بنية الشرط if فتناسب المجالات المفتوحة أكثر.

أنشئ الأداتين التاليتين، واملأ صندوق اللائحة listBox1 ببعض العناصر:



إذا نقرت على أحد الزرين فلن يحصل شيء؛ إذ ليس هناك عنصرًا محددًا. اختر العنصر c مثلًا وانقر زر UP بالزر الأيسر للفأرة ثم الزر DOWN بالزر الأيمن، ولاحظ (اليسار ثم اليمين):





يمكنك تطوير الأداة أيضًا بجعلها تنقل العناصر من خلال الضغط على أزرار لوحة المفاتيح، فمثلًا: يُنقل العنصر للأعلى خطوة واحدة عند تحديده والضغط على {UP} + {SHIFT}، وينقل لأعلى القائمة عند الضغط على {UP} + {SHIFT} + {CTRL}. وهكذا.

كما يمكنك تطوير الأداة بجعل المبرمج يختار هل هي أفقية أم شاقولية.

ListBoxTransfer

لا شك أنك قابلت كثيرًا أداتي لائحة ListBox ونقلت بيانات إحداها إلى الأخرى (مثل معالجات تثبيت بعض البرامج التي تطلب منك نقل المكونات التي تود تثبيتها من لائحة أولى فيها كل المكونات المتاحة إلى لائحة ثانية فيها المكونات التي تود تثبيتها). تشبه كثيرًا هذه الأداة الأداة التي سبقتها، إلا أني تعمدت تغيير الأسلوب قليلًا، فعوضًا عن استخدام طريقة واحدة لنقل العناصر (خطوة واحدة إلى بداية أو نهاية القائمة)، جزأت المسألة، وخصصت لكل حالة طريقة.



```
using System;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class ListBoxTransfer : Eng27UserControl {
        #region Local Variables
        ButtonFlat button1, button2, button3, button4;
        ListBox listbox1, listbox2;
        #endregion


        #region Constructros
        ///
        public ListBoxTransfer() {
            this.Width = 200;
            this.Height = 30;

            button1 = new ButtonFlat
            {
                Dock = DockStyle.Left,
                Width = 40,
                ButtonBorderStyle = buttonborderstyle,
                Text = "<"
            };

            button2 = new ButtonFlat
            {
                Dock = DockStyle.Left,
                Width = 40,
                ButtonBorderStyle = buttonborderstyle,
                Text = "<<"
            };

            button3 = new ButtonFlat
            {
                Dock = DockStyle.Right,
                Width = 40,
                ButtonBorderStyle = buttonborderstyle,
                Text = ">"
            };
```

```
        button4 = new ButtonFlat
        {
            Dock = DockStyle.Right,
            Width = 40,
            ButtonBorderStyle = buttonborderstyle,
            Text = ">>"
        };

        this.Controls.Add(button1);
        this.Controls.Add(button2);
        this.Controls.Add(button3);
        this.Controls.Add(button4);

        button1.MouseDown += button1_MouseDown;
        button3.MouseDown += button3_MouseDown;
        button2.MouseDown += button2_MouseDown;
        button4.MouseDown += button4_MouseDown;
    }
    #endregion


    #region Properties
    private bool autobuttonsheight = false;
    ///
    public bool AutoButtonsHeight
    {
        get { return autobuttonsheight; }
        set
        {
            autobuttonsheight = value;
            if (value) this.ButtonWidth = this.Width / 4;
        }
    }

    private int buttonWidth = 50;
    ///
    public int ButtonWidth
    {
        get { return buttonWidth; }
        set
        {
            buttonWidth = value;
            button1.Width = button2.Width = value;
            button3.Width = button4.Width = value;
        }
    }

    private ButtonBorderStyle buttonborderstyle = ButtonBorderStyle.Solid;
    ///
    public ButtonBorderStyle ButtonBorderStyle
    {
        get { return buttonborderstyle; }
        set
        {
            buttonborderstyle = value;
            button1.ButtonBorderStyle = value;
            button2.ButtonBorderStyle = value; }
    }
```

```
        private bool movetofirstandlast = true;
        ///
        public bool MoveToFirstAndLast
        {
            get { return movetofirstandlast; }
            set { movetofirstandlast = value; }
        }

        private bool clearaftermove = true;
        ///
        public bool ClearAfterMove
        {
            get { return clearaftermove; }
            set { clearaftermove = value; }
        }

        ///
        public ListBox ListBox1
        {
            get { return listbox1; }
            set { listbox1 = value; }
        }

        ///
        public ListBox ListBox2
        {
            get { return listbox2; }
            set { listbox2 = value; }
        }

        public ThemeManager ThemeManager ...
        public Style_e Style ...
        public StyleDark StyleDark ...
        public StyleLight StyleLight ...
        public Theming_e Theming ...
        #endregion


        #region Methods
        bool ListBoxesNotNull()
        {
            if (listbox1 == null || listbox2 == null)
                return false;
            else
                return true;
        }

        ///
        public void MoveItem(ListBox lstFrom, ListBox lstTo)
        {
            if (ListBoxesNotNull())
            {
                if (lstFrom.SelectedIndex >= 0)
                {
                    lstTo.Items.Add(lstFrom.Items[lstFrom.SelectedIndex]);
                    if (clearaftermove)
                        lstFrom.Items.RemoveAt(lstFrom.SelectedIndex);
                    lstTo.SelectedIndex = lstTo.Items.Count - 1; } } }
```

```
        ///
        public void MoveAllItems(ListBox lstFrom, ListBox lstTo)
        {
            if (ListBoxesNotNull())
                if (lstFrom.Items.Count > 0)
                {
                    lstTo.Items.AddRange(lstFrom.Items);
                    if (clearaftermove)
                        lstFrom.Items.Clear();
                }
        }

        protected override void SetBoundsCore
            (int x, int y, int width, int height, BoundsSpecified specified)
        {
            if (autobuttonsheight)
            {
                this.ButtonWidth = width / 4;
            }
            else
            {
                if (width < 4 * buttonWidth)
                    width = 4 * buttonWidth;
            }
            base.SetBoundsCore(x, y, width, height, specified);
        }
        #endregion


        #region Events
        protected override void OnSizeChanged(EventArgs e)
        {
            if (this.Width < 4 * buttonWidth)
                this.Width = 4 * buttonWidth;
        }

        void button4_MouseDown(object sender, MouseEventArgs e) {
            MoveAllItems(listbox1, listbox2);
        }

        void button3_MouseDown(object sender, MouseEventArgs e) {
            MoveItem(listbox1, listbox2);
        }

        void button2_MouseDown(object sender, MouseEventArgs e) {
            MoveAllItems(listbox2, listbox1);
        }

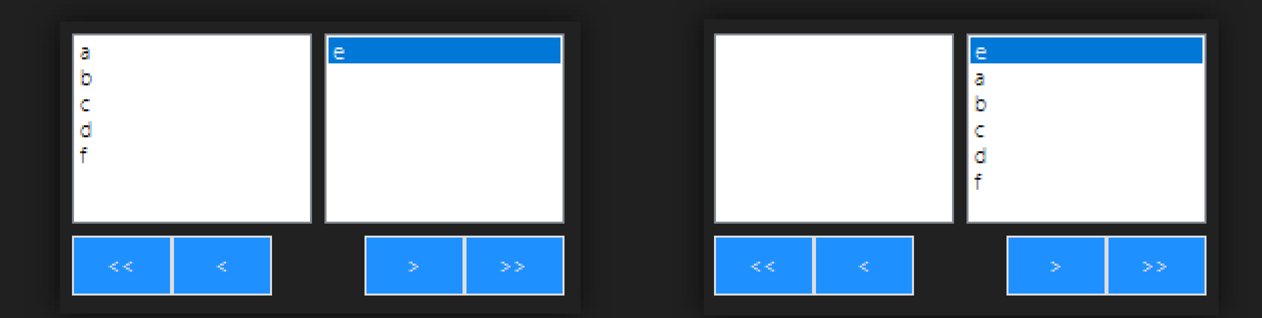
        void button1_MouseDown(object sender, MouseEventArgs e) {
            MoveItem(listbox2, listbox1);
        }

        void StyleLight_StyleChanged(object sender, System.EventArgs e) ...
        void StyleDark_StyleChanged(object sender, System.EventArgs e) ...
        void thememanager_ThemeChanged(object sender, ThemingEventArgs e) ...
        #endregion
    }
}
```

أنشئ الأدوات الثلاث التالية واضبط الخصائص ListBox1 وListBox2، وأضف بعض العناصر إلى صناديق اللوائح كما يلي:



اختر أحد العناصر من اللائحة الأولى وانقله إلى الثانية، ثم انقل جميع العناصر، ولاحظ النتيجة (من اليسار لليمين):



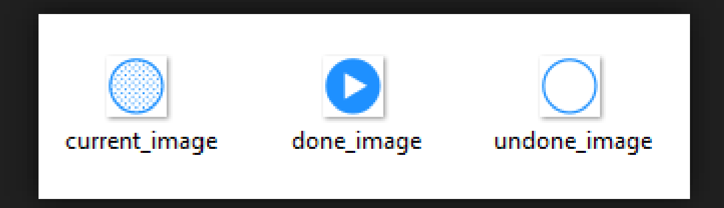
جميع اقتراحات التطوير الموجودة في الفقرة السابقة يمكن تطبيقها في هذه الفقرة.

StepsProgress

تشابه هذه الأداة أداة ProgressBar في أنهما يعرضان تقدّم المستخدم، وتختلف عنها في كونها محدودة الخطوات، وذات طبيعة صورية (لكل خطوة صورة تصفها).

اعتمدت على الفئة Eng27UserControl لتصميم الأداة، مع أنه من الممكن أيضًا تصميمها اعتمادًا على الرسم (بالاعتماد على الحدث OnPaint، والذي سنناقش بعض الأدوات التي سنصممها وفقه لاحقًا).

قم بإضافة ثلاث صور لمصادر المشروع Resources واجعلها مضمنةً (اضبط الخاصية Persistance لهذه الصور على Embedded in .resx). على سبيل المثال قمت بإضافة هذه الصور (سمّ الصور كما يلي):





```
using Eng27.Components;
using Eng27.Enums;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class StepsProgress : Eng27Control
    {
        #region Local Variables
        int steps, currentstep;
        int w, h;
        Image stepdone, stepcurrent, stepundone;
        Color linecolor;
        PictureBox[] pb;
        Panel p;
        #endregion


        #region
        ///
        public StepsProgress()
        {
            w = 300; h = 30;
            Size = new Size(w, h);
            steps = 5;
            currentstep = 0;
            stepdone = Properties.Resources.done_image;
            stepcurrent = Properties.Resources.current_image;
            stepundone = Properties.Resources.undone_image;
            linecolor = Color.Black;
            this.CreateSteps();
        }
        #endregion
```

```
#region Properties
///
public int Steps
{
    get { return steps; }
    set { steps = value; this.CreateSteps(); }
}

///
public int CurrentStep
{
    get { return currentstep; }
    set
    {
        if (value == 0)
        {
            currentstep = value;
            this.CreateSteps();
        }

        else if (value == steps + 1)
        {
            currentstep = value;
            this.FinishSteps();
        }

        else if (value <= steps)
        {
            currentstep = value; this.RefreshSteps();
        }
    }
}

///
public Image StepDone
{
    get { return stepdone; }
    set { stepdone = value; this.RefreshSteps(); }
}

///
public Image StepCurrent
{
    get { return stepcurrent; }
    set { stepcurrent = value; this.RefreshSteps(); }
}

///
public Image StepUnDone
{
    get { return stepundone; }
    set { stepundone = value; this.RefreshSteps(); }
}

///
public Color LineColor
{
    get { return linecolor; }
    set { linecolor = value; this.RefreshSteps(); }
}
```

```
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public ThemeManager ThemeManager { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Theming_e Theming { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleDark StyleDark { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleLight StyleLight { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Style_e Style { get; set; }
        #endregion


        #region Methods
        protected override void SetBoundsCore
            (int x, int y, int width, int height, BoundsSpecified specified)
        {
            base.SetBoundsCore(x, y, width, h, BoundsSpecified.None);
            w = width;
            this.RefreshSteps();
        }

        void CreateSteps()
        {
            Controls.Clear();
            pb = new PictureBox[steps];
            p = new Panel();
            int current = 0;
            for (int i = 0; i < steps; i++)
            {
                pb[i] = new PictureBox();
                pb[i].Size = new Size(h, h);
                pb[i].BackgroundImageLayout = ImageLayout.Zoom;
                pb[i].Left = current;
                current += h + (int)((w - steps * h) / (steps - 1));
                pb[i].BackgroundImage = stepundone;
                pb[i].BackColor = Color.Transparent;
                pb[i].Click += StepClick;
                pb[i].Cursor = Cursors.Hand;
                this.Controls.Add(pb[i]);
            }
            p.Left = (int)(h / 2);
            p.Top = (int)(h / 2);
            p.Height = 3;
            p.Width = w - h;
            p.BackColor = linecolor;
            p.SendToBack();
            this.Controls.Add(p);

            currentstep = 0;
        }
```

```
        void RefreshSteps()
        {
            if (currentstep > 0)
            {
                this.Controls.Clear();
                pb = new PictureBox[steps];
                int current = 0;
                for (int i = 0; i < steps; i++)
                {
                    pb[i] = new PictureBox();
                    pb[i].Size = new Size(h, h);
                    pb[i].BackgroundImageLayout = ImageLayout.Zoom;
                    pb[i].Left = current;
                    current += h + (int)((w - steps * h) / (steps - 1));
                    if (i < currentstep - 1)
                        pb[i].BackgroundImage = stepdone;
                    if (i >= currentstep & i < steps)
                        pb[i].BackgroundImage = stepundone;
                    pb[i].Click += StepClick;
                    pb[i].Cursor = Cursors.Hand;
                    this.Controls.Add(pb[i]);
                }
                pb[currentstep - 1].BackgroundImage = stepcurrent;

                p = new Panel();

                p.Left = (int)(h / 2);
                p.Top = (int)(h / 2);
                p.Height = 3;
                p.Width = w - h;
                p.BackColor = linecolor;
                p.SendToBack();
                this.Controls.Add(p);
            }
            else
                this.CreateSteps();
        }

        void FinishSteps()
        {
            this.Controls.Clear();
            pb = new PictureBox[steps];
            int current = 0;
            for (int i = 0; i < steps; i++)
            {
                pb[i] = new PictureBox();
                pb[i].Size = new Size(h, h);
                pb[i].BackgroundImageLayout = ImageLayout.Zoom;
                pb[i].Left = current;
                current += h + (int)((w - steps * h) / (steps - 1));
                pb[i].BackgroundImage = stepdone;
                pb[i].Click += StepClick;
                pb[i].Cursor = Cursors.Hand;
                this.Controls.Add(pb[i]);
            }

            p = new Panel();

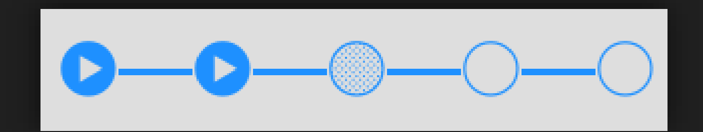
            p.Left = (int)(h / 2);
            p.Top = (int)(h / 2);
```

```
            p.Height = 3;
            p.Width = w - h;
            p.BackColor = linecolor;
            p.SendToBack();
            this.Controls.Add(p);
        }
        #endregion

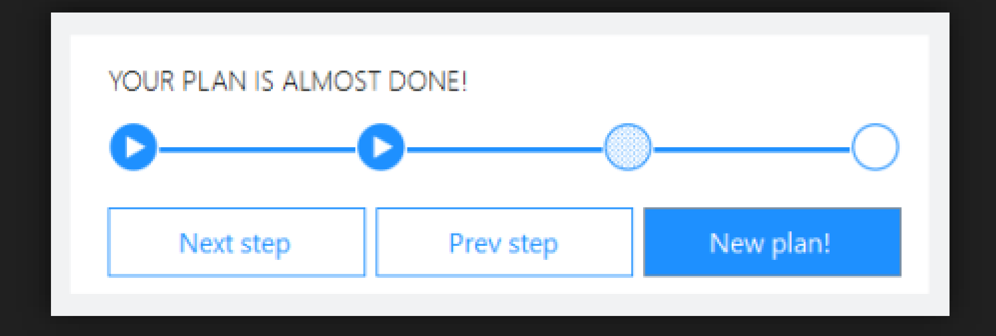

        #region Events
        protected override void OnParentBackColorChanged(EventArgs e)
        {
            this.BackColor = Parent.BackColor;
        }

        void StepClick(object sender, EventArgs e)
        {
            PictureBox p = (PictureBox)sender;
            this.CurrentStep = ((Control)p).TabIndex + 1;
        }
        #endregion
    }
}
```

أنشئ نسخة من الأداة، واضبط لون الخط فيها على DodgerBlue، ثم شغل المشروع واختر إحدى الخطوات:



الفصل العاشر سيحوي تطبيقًا يضم هذه الأداة:



يمكنك تطوير الأداة بجعلها تدعم مدير السمات ThemeManager وجعلها تقبل الوضع الشاقولي أو الأفقي، كما يمكنك – بتصميمها اعتمادًا على الرسومات (من خلال الحدث OnPaint) – الوصول لإمكانيات أكبر.

طوّر أداة – على مبدأ أدوات التعامل مع عناصر صناديق اللوائح – تقوم بالتعامل مع هذه الأداة، بحيث تأخذها ككائن لأحد خصائصها، لتنتقل للخطوة التالية أو السابقة، أو الأولى أو الأخيرة.

أدوات ذات طبيعة متحركة AnimatedControls

ButtonAnimated



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Exceptions;
using Eng27.Interfaces;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class ButtonAnimated : Eng27ButtonBase, IEng27Animation
    {
        #region Local Variables
        ButtonFlat button;
        Style CurrentStyle = new Style();
        Timer timerin, timerout;
        bool in_control = false;
        bool in_button = false;
        #endregion


        #region Constructors
        ///
        public ButtonAnimated()
        {
            this.FlatAppearance.BorderSize = 1;
            this.Height = 100;
            this.CurrentStyle = styledark;

            button = new ButtonFlat
            {
                Dock = DockStyle.Bottom,
                Height = 5,
                ButtonBorderStyle = ButtonBorderStyle.Solid,
```

```
            BorderSize = 1,
            BorderColor = Color.FromArgb(tranparency, CurrentStyle.ForeColor),
            ForeColor = CurrentStyle.ForeColor,
            Style = this.Style,
            StyleDark = styledark,
            StyleLight = stylelight
        };

        timerin = new Timer();
        timerout = new Timer();
        this.Controls.Add(button);

        this.MouseEnter += AnimatedButton_MouseEnter;
        this.MouseLeave += AnimatedButton_MouseLeave;

        timerin.Interval = interval;
        timerin.Tick += timerin_Tick;

        timerout.Interval = interval;
        timerout.Tick += timerout_Tick;

        button.MouseEnter += button_MouseEnter;
        button.MouseLeave += button_MouseLeave;
        button.Click += button_Click;

        this.StyleDark.StyleChanged += StyleDark_StyleChanged;
        this.StyleLight.StyleChanged += StyleLight_StyleChanged;
    }
    #endregion


    #region Properties
    public Style_e Style ...
    public StyleDark StyleDark ...
    public StyleLight StyleLight ...
    public ThemeManager ThemeManager ...
    public Theming_e Theming ...

    // هذه الخاصية للقراءة فقط
    private bool isanimating = false;
    /// <summary>
    /// Gets a value indecate that the control is animating.
    /// </summary>
    [Browsable(false)]
    public bool IsAnimating
    {
        get { return isanimating; }
    }

    private bool enableanimation = true;
    ///
    public bool EnableAnimation
    {
        get { return enableanimation; }
        set { enableanimation = value; }
    }
```

```
        private int steps = 5;
        ///
        public int Steps
        {
            get { return steps; }
            set { steps = value; }
        }

        private double leavesteps = 1.5;
        ///
        public double LeaveSteps
        {
            get { return leavesteps; }
            set { leavesteps = value; }
        }

        private byte tranparency = 150;
        ///
        public byte Tranparency
        {
            get { return tranparency; }
            set
            {
                tranparency = value;
                button.BackColor = Color.FromArgb(tranparency,
                    CurrentStyle.BackColor);
            }
        }

        private int delay = 333;
        ///
        public int Delay
        {
            get { return delay; }
            set { delay = value; }
        }

        private int animatedButtonHeight = 75;
        ///
        public int AnimatedButtonHeight
        {
            get { return animatedButtonHeight; }
            set
            {
                if (value > 5 && value < 100) animatedButtonHeight = value;
                else throw new AnimatedButtonHeightException(value);
            }
        }

        int interval = 10;
        ///
        public int Interval
        {
            get { return interval; }
            set {
                interval = value;
                timerin.Interval = value;
                timerout.Interval = value;
            }
        }
```

```
        private string button_text = "Button Text";
        ///
        public string ButtonText
        {
            get { return button_text; }
            set { button_text = value; button.Text = button_text; }
        }
        #endregion


        #region Methods
        void ChangeStyle()
        {
            switch (style)
            {
                case Style_e.Dark:
                    this.BackColor = styledark.BackColor;
                    this.ForeColor = styledark.ForeColor;
                    this.Font = styledark.Font;
                    this.CurrentStyle = styledark;
                    break;

                case Style_e.Light:
                    this.BackColor = stylelight.BackColor;
                    this.ForeColor = stylelight.ForeColor;
                    this.Font = stylelight.Font;
                    this.CurrentStyle = stylelight;
                    break;

                default:
                    break;
            }
            button.BorderColor = Color.FromArgb(tranparency, this.ForeColor);
            button.Style = style;
        }
        #endregion


        #region Events
        void button_MouseLeave(object sender, EventArgs e)
        {
            in_button = false;
            timerin.Enabled = false;
            timerout.Enabled = true;
        }

        void button_MouseEnter(object sender, EventArgs e)
        {
            in_button = true;
            timerout.Enabled = false;
            timerin.Enabled = true;
            isanimating = true;
        }

        void button_Click(object sender, EventArgs e)
        {
            this.OnButtonClick(e);
        }
```

```
void timerout_Tick(object sender, EventArgs e)
{
    if (enableanimation)
    {
        if (button.Height > 5)
        {
            if (!in_control && !in_button)
                button.Height -= (int)(steps * leavesteps);
        }
        else
        {
            button.Height = 5;
            timerout.Enabled = false;
            isanimating = false;
        }
    }
}

void AnimatedButton_MouseLeave(object sender, EventArgs e)
{
    in_control = false;
    timerin.Enabled = false;
    timerout.Enabled = true;
}

void timerin_Tick(object sender, EventArgs e)
{
    if (enableanimation)
        if (button.Height <= animatedButtonHeight * Height / 100)
        {
            if (in_control || in_button)
                button.Height += steps;
        }
}

private void AnimatedButton_MouseEnter(object sender, EventArgs e)
{
    in_control = true;
    Task.Delay(delay).Wait();

    timerout.Enabled = false;
    timerin.Enabled = true;
}

void StyleLight_StyleChanged(object sender, System.EventArgs e)
{
    this.ChangeStyle();
    this.OnThemeChanged(new ThemingEventArgs(ThemingArgs_e.StyleChanged));
}

void StyleDark_StyleChanged(object sender, System.EventArgs e)
{
    this.ChangeStyle();
    this.OnThemeChanged(new ThemingEventArgs(ThemingArgs_e.StyleChanged));
}

void thememanager_ThemeChanged(object sender, ThemingEventArgs e) ...
#endregion
```

```
        #region Custom Events
        ///
        public event EventHandler ButtonClick;
        ///
        protected virtual void OnButtonClick(EventArgs e)
        {
            if (ButtonClick != null)
                ButtonClick.Invoke(this, e);
        }

        ///
        public delegate void ThemingEventHandler(object send, ThemingEventArgs e);
        ///
        public event ThemingEventHandler ThemeChanged;
        ///
        protected virtual void OnThemeChanged(ThemingEventArgs e)
        {
            if (ThemeChanged != null)
                ThemeChanged.Invoke(this, e);
        }
        #endregion
    }
}
```

من الأمور التي عليك تطويرها، هو إضفاء الشفافية على الأداة عند ضبط المظهر؛ إذ إنها لن تظهر بهذا الكود.

الأداة فيها أداة زر من النوع ButtonFlat، يتم ضبط ارتفاع هذه الأداة من خلال الخاصية AnimatedButtonHeight، وهي نسبة، تتراوح بين 5% و100% من ارتفاع الأداة، فإذا تم إدخال قيمة خارج هذا المجال حدث استثناء AnimatedButtonHeightException:



```
using System;

namespace Eng27.Exceptions
{
    ///
    public class AnimatedButtonHeightException :Exception
    {
        ///
        public AnimatedButtonHeightException(int value)
            : base("Height cannot be more than 100% or less than 5%")
        {
            this.Value = value;
        }

        public int Value { get; private set; }
    }
}
```

كما هو الحال مع الأحداث، إذ إنك تنشئ مفوّضًا خاصًّا لتمرير بيانات مع هذه الأحداث؛ فإنك تفعل ذات الشيء مع الاستثناءات، فإذا لم تكن هناك بيانات تحتاج لتمريرها يمكنك الاعتماد على الفئة Exception مباشرة، وذلك عند إطلاق الاستثناء باستخدام الكلمة throw.

فلو حدث استثناء، فإن للمبرمجين فرصة للتحكم به:

```
try
{
    animated_button.AnimatedButtonHeight = 0;
}
catch (AnimatedButtonHeightException ex)
{
    MessageBox.Show("You cannot set Value to: " + ex.Value);
}
```

You cannot set Value to: 0
OK

ستحتاج لتضمين مجال الأسماء Eng27.Exceptions لتستطيع الوصول للاستثناءات.

الخصائص للقراءة فقط يجب ألا تظهر في نافذة الخصائص، لذلك عليك توصيفها بالمواصفة Browsable وتمرير القيمة false لها.

ButtonDoubleAnimated

تشبه هذه الأداة أداة ButtonAnimated، إلا أن فيها زرين، عند مرور مؤشر الفأرة على أحد الزرين فإنه يكبر؛ مما يجعل تركيز المستخدم وانتباهه عليه.

```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System;
using System.ComponentModel;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class ButtonDoubleAnimated : Eng27UserControl, IEng27Animation
    {
```

```
#region Local Variables
ButtonFlat button1, button2;
Timer timer1, timer2;
Timer timerleave;
bool button1in = false, button2in = false;
bool buttonout = false;
#endregion


#region Constructors
///
public ButtonDoubleAnimated()
{
    button1 = new ButtonFlat
    {
        Name = "button1",
        Dock = DockStyle.Top,
        ButtonBorderStyle = buttonborderstyle,
    };

    Controls.Add(button1);

    button2 = new ButtonFlat
    {
        Name = "button2",
        Dock = DockStyle.Fill,
        ButtonBorderStyle = buttonborderstyle,
    };

    Controls.Add(button2);

    button1.Height = (int)(.5 * Height);
    button2.BringToFront();

    ContextMenuChanged += ButtonDoubleAnimated_ContextMenuChanged;
    ContextMenuStripChanged+=ButtonDoubleAnimated_ContextMenuStripChanged;

    button1.Click += button1_Click;
    button2.Click += button2_Click;

    timer1 = new Timer();
    timer2 = new Timer();
    timerleave = new Timer();

    timer1.Interval = interval;
    timer2.Interval = interval;
    timerleave.Interval = (int)(interval * outInterval);

    timer1.Tick += timer1_Tick;
    timer2.Tick += timer2_Tick;
    timerleave.Tick += timerleave_Tick;
    button1.MouseEnter += button1_MouseEnter;
    button1.MouseLeave += button_MouseLeave;

    button2.MouseEnter += button2_MouseEnter;
    button2.MouseLeave += button_MouseLeave;
}
#endregion
```

```
#region Properties
public Style_e Style ...
public StyleDark StyleDark ...
public StyleLight StyleLight ...
public ThemeManager ThemeManager ...
public Theming_e Theming ...

private ButtonBorderStyle buttonborderstyle = ButtonBorderStyle.Solid;
///
public ButtonBorderStyle ButtonBorderStyle
{
    get { return buttonborderstyle; }
    set
    {
        buttonborderstyle = value;
        button1.ButtonBorderStyle = value;
        button2.ButtonBorderStyle = value;
    }
}

private bool isanimating = false;
///
public bool IsAnimating
{
    get { return isanimating; }
}

private int steps = 5;
///
public int Steps
{
    get { return steps; }
    set { steps = value; }
}

private int delay = 333;
///
public int Delay
{
    get { return delay; }
    set { delay = value; }
}

int interval = 20;
///
public int Interval
{
    get { return interval; }
    set
    {
        interval = value;
        timer1.Interval = value;
        timer2.Interval = value;
    }
}
```

```
        private double outInterval = 0.5;
        ///
        public double OutInterval
        {
            get { return outInterval; }
            set
            {
                outInterval = value;
                timerleave.Interval = (int)(value * interval);
            }
        }

        private string text1;
        ///
        public string Text1
        {
            get { return text1; }
            set { text1 = value; button1.Text = value; }
        }

        private string text2;
        ///
        public string Text2
        {
            get { return text2; }
            set { text2 = value; button2.Text = value; }
        }
        #endregion


        #region Methods
        void ChangeStyle()
        {
            switch (style)
            {
                case Style_e.Dark:
                    this.BackColor = styledark.BackColor;
                    this.ForeColor = styledark.ForeColor;
                    this.Font = styledark.Font;
                    break;

                case Style_e.Light:
                    this.BackColor = stylelight.ForeColor;
                    this.ForeColor = stylelight.BackColor;
                    this.Font = stylelight.Font;
                    break;

                default:
                    break;
            }

            button1.BackColor = this.BackColor;
            button1.ForeColor = this.ForeColor;
            button1.Font = this.Font;

            button2.BackColor = this.BackColor;
            button2.ForeColor = this.ForeColor;
            button2.Font = this.Font;
```

```
        if (thememanager != null)
        {
            if (thememanager.UserBorderSettings)
            {
                button1.BorderColor = thememanager.BorderColor;
                button2.BorderColor = thememanager.BorderColor;
            }
            else
            {
                button1.BorderColor = this.ForeColor;
                button2.BorderColor = this.ForeColor;
            }
            button1.BorderSize = thememanager.BorderSize;
            button2.BorderSize = thememanager.BorderSize;
        }
        else
        {
            button1.BorderColor = this.ForeColor;
            button2.BorderColor = this.ForeColor;
        }
    }
    #endregion


    #region Events
    void timerleave_Tick(object sender, EventArgs e)
    {
        if (!buttonout)
        {
            if (!button1in && !button2in)
            {
                if (button1.Height < Height * .5)
                    button1.Height += steps;
                if (button2.Height < Height * .5)
                    button1.Height -= steps;
            }
            else
            {
                timerleave.Enabled = false;
                buttonout = true;
                button1.Height = (int)(Height * .5);
                isanimating = false;
            }
        }
    }

    void button_MouseLeave(object sender, EventArgs e)
    {
        button1in = button2in = false; buttonout = false;
        Task.Delay(delay).Wait();
        timerleave.Enabled = true;
        OnButtonLeave(new ButtonDoubleAnimatedEventArgs((ButtonFlat)sender));
    }

    void button2_MouseEnter(object sender, EventArgs e) {
        button2in = true;
        timer2.Enabled = true;
        this.OnButton2Enter(e);
        isanimating = true;
    }
```

```
void button1_MouseEnter(object sender, EventArgs e)
{
    button1in = true;
    timer1.Enabled = true;
    this.OnButton1Enter(e);
    isanimating = true;
}

void timer2_Tick(object sender, EventArgs e)
{
    if (button2.Height < Height * .75)
        button1.Height -= steps;
    else
        timer2.Enabled = false;
}

void timer1_Tick(object sender, EventArgs e)
{
    if (button1.Height < Height * .75)
        button1.Height += steps;
    else
        timer1.Enabled = false;
}

void ButtonDoubleAnimated_ContextMenuStripChanged(object sen, EventArgs e)
{
    foreach (Control c in Controls)
        c.ContextMenuStrip = this.ContextMenuStrip;

}

void ButtonDoubleAnimated_ContextMenuChanged(object sender, EventArgs e)
{
    foreach (Control c in Controls)
    {
        c.ContextMenu = this.ContextMenu;
    }
}

void button1_Click(object sender, EventArgs e) {
    this.OnButton1Click(e);
}

void button2_Click(object sender, EventArgs e) {
    this.OnButton2Click(e);
}

private void thememanager_ThemeChanged ...
#endregion


#region Custom Events
///
public event EventHandler Button1Click;
///
protected virtual void OnButton1Click(EventArgs e)
{
    if (Button1Click != null)
        Button1Click.Invoke(button1, e);
}
```

```
        ///
        public event EventHandler Button2Click;
        ///
        protected virtual void OnButton2Click(EventArgs e)
        {
            if (Button2Click != null)
                Button2Click.Invoke(button2, e);
        }

        ///
        public event EventHandler Button1Enter;
        ///
        protected virtual void OnButton1Enter(EventArgs e)
        {
            if (Button1Enter != null)
                Button1Enter.Invoke(button1, e);
        }

        ///
        public event EventHandler Button2Enter;
        ///
        protected virtual void OnButton2Enter(EventArgs e)
        {
            if (Button2Enter != null)
                Button2Enter.Invoke(button2, e);
        }

        ///
        public delegate void ButtonDoubleAnimatedEventHandler
            (object source, ButtonDoubleAnimatedEventArgs e);
        ///
        public event ButtonDoubleAnimatedEventHandler ButtonLeave;
        ///
        protected virtual void OnButtonLeave(ButtonDoubleAnimatedEventArgs e)
        {
            if (ButtonLeave != null)
                ButtonLeave.Invoke(this, e);
        }

        ///
        public delegate void ThemingEventHandler(object send, ThemingEventArgs e);
        ///
        public event ThemingEventHandler ThemeChanged;
        ///
        protected virtual void OnThemeChanged(ThemingEventArgs e)
        {
            if (ThemeChanged != null)
                ThemeChanged.Invoke(this, e);
        }
        #endregion
    }
}
```

بالإضافة للحدث ThemeChanged الذي سبق مناقشته مع الأداة ThemeManager، فإنه لهذه الأداة حدث خاص يحتاج مفوّضًا خاصًا، وهو الحدث ButtonLeave، فهو يحمل

في طياته الزر الذي تمت مغادرته (على اعتبار أن الأداة تحوي زرين)؛ لذلك فلن يغنيك استخدام الفئة EventHandler، إلا إذا أنشأت حدثين، كل واحد منهما يتعلق بزر، كما في الأحداث الأخرى.



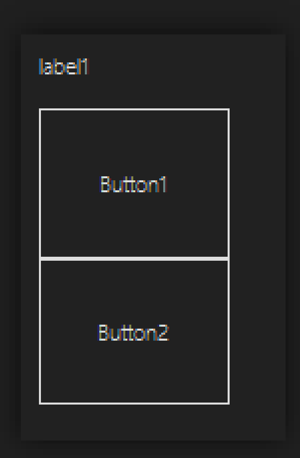
```
using Eng27.UI;
using System;

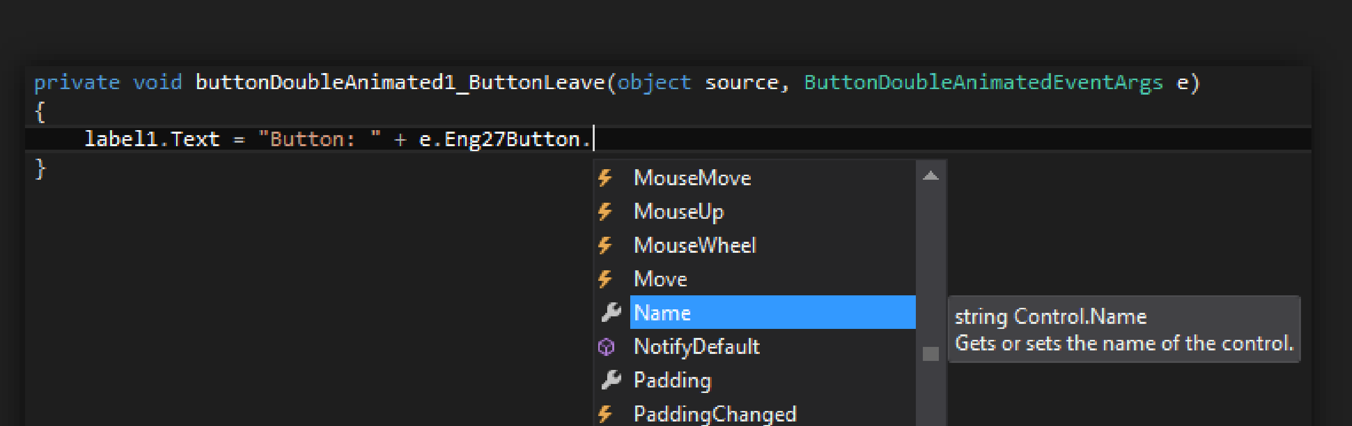
namespace Eng27
{
    ///
    public class ButtonDoubleAnimatedEventArgs : EventArgs
    {
        private ButtonFlat button;
        ///
        public ButtonDoubleAnimatedEventArgs(ButtonFlat b)
        {
            button = b;
        }

        ///
        public string GetControlName()
        {
            return button.Name;
        }

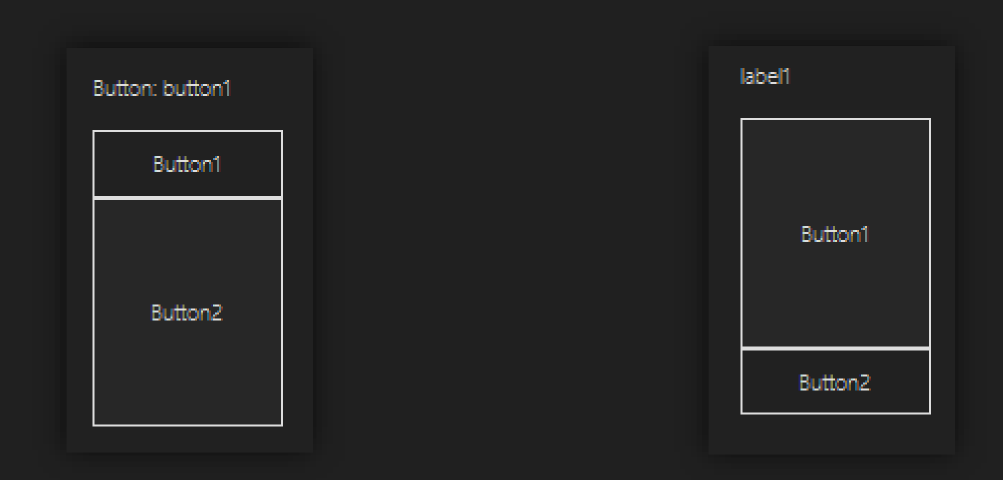
        ///
        public ButtonFlat Eng27Button { get { return button; } }
    }
}
```

أنشئ نسخة من الأداة، وأنشئ أداة عنوان Label، واعتمد على حدث مغادرة الزر لمعرفة الزر:





شغل البرنامج وضع مؤشر الفأرة مرة على الزر الأول ومرة على الزر الثاني:



ButtonCircularAnimated

هذا الزر دائري، ويتميز بأن حوافه تُرسم عند اقتراب مؤشر الفأرة من الأداة، يمكنك تطوير الأداة هذه بجعل حوافها ترسم عند النقر على الأداة مثلًا أو حدوث حدث آخر.

```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;
```

```
namespace Eng27.UI
{
    ///
    public class ButtonCircularAnimated
        : Eng27Control, IEng27Radial, IEng27Animation
    {
        #region Local Variables
        Timer timer = new Timer();
        Label label;
        float ang = 0;
        #endregion


        #region Constructors
        ///
        public ButtonCircularAnimated()
        {
            this.MouseEnter += ButtonCircularAnimated_MouseEnter;

            this.Size = new Size(50, 50);

            label = new Label
            {
                AutoSize = false,
                Dock = DockStyle.Fill,
                TextAlign = ContentAlignment.MiddleCenter,
                BackColor = Color.Transparent,
                Text = this.Text
            };

            label.MouseEnter += label_MouseEnter;
            label.MouseDown += label_MouseDown;
            label.MouseUp += label_MouseUp;
            label.Click += label_Click;
            this.Controls.Add(label);

            timer.Interval = interval;
            timer.Tick += timer_Tick;

            bordercolor = this.ForeColor;
        }
        #endregion


        #region Properties
        public Style_e Style ...
        public StyleDark StyleDark ...
        public StyleLight StyleLight ...
        public ThemeManager ThemeManager ...
        public Theming_e Theming ...

        private int steps = 7;
        ///
        public int Steps
        {
            get { return steps; }
            set { steps = value; }
        }
```

```
        private int interval = 10;
        ///
        public int Interval
        {
            get { return interval; }
            set { interval = value; timer.Interval = value; }
        }

        private bool isanimating = false;
        ///
        public bool IsAnimating
        {
            get { return isanimating; }
        }

        private int bordersize = 2;
        ///
        public int BorderSize
        {
            get { return bordersize; }
            set { bordersize = value; }
        }

        private Color bordercolor;
        ///
        public Color BorderColor
        {
            get { return bordercolor; }
            set { bordercolor = value; }
        }

        ///
        public new string Text
        {
            get { return base.Text; }
            set { base.Text = label.Text = value; }
        }

        private int radius;
        ///
        public int Radius
        {
            get { return radius; }
            set { radius = value; }
        }
        #endregion


        #region Methods
        void ChangeStyle()
        {
            switch (style)
            {
                case Style_e.Dark:
                    this.BackColor = styledark.BackColor;
                    this.ForeColor = styledark.ForeColor;
                    this.Font = styledark.Font;
                    break;
```

```
            case Style_e.Light:
                this.BackColor = stylelight.ForeColor;
                this.ForeColor = stylelight.BackColor;
                this.Font = stylelight.Font;
                break;

            default:
                break;
        }

        if (thememanager != null)
        {
            if (thememanager.UserBorderSettings)
                bordercolor = thememanager.BorderColor;
            else
                bordercolor = this.ForeColor;
            bordersize = thememanager.BorderSize;
        }

        else
            bordercolor = this.ForeColor;
    }
    #endregion


    #region Events
    void label_MouseUp(object sender, MouseEventArgs e)
    {
        this.OnMouseUp(e);
    }

    void label_MouseDown(object sender, MouseEventArgs e)
    {
        this.OnMouseDown(e);
    }

    void label_MouseEnter(object sender, EventArgs e)
    {
        this.OnMouseEnter(e);
    }

    void label_Click(object sender, EventArgs e)
    {
        this.OnClick(e);
    }

    void timer_Tick(object sender, EventArgs e)
    {
        ang += steps;
        if (360 - ang <= steps)
        {
            isanimating = false;
            ang = 360;
            timer.Enabled = false;
        }
        Invalidate();
    }
```

```
protected override void OnPaint(PaintEventArgs e)
{
    e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;
    if (!isanimating)
        e.Graphics.DrawEllipse(
            new Pen(bordercolor, bordersize),
            new Rectangle(bordersize / 2, bordersize / 2,
                this.Width - 3 * bordersize / 2,
                this.Height - 3 * bordersize / 2));

    e.Graphics.DrawArc(
        new Pen(bordercolor, bordersize),
        new Rectangle(bordersize / 2, bordersize / 2,
            this.Width - 3 * bordersize / 2,
            this.Height - 3 * bordersize / 2), 0, ang);
}

void ButtonCircularAnimated_MouseEnter(object sender, EventArgs e)
{
    if (!isanimating)
    {
        this.CreateGraphics().Clear(this.BackColor);
        ang = 0;
        isanimating = true;
        timer.Enabled = true;
    }
}

protected override void SetBoundsCore
    (int x, int y, int width, int height, BoundsSpecified specified)
{
    width = height;
    base.SetBoundsCore(x, y, width, height, specified);
}

protected override void OnSizeChanged(EventArgs e)
{
    this.Width = this.Height;
    base.OnSizeChanged(e);
}

private void thememanager_ThemeChanged(object sender, ThemingEventArgs e)
{
    switch (e.Theming)
    {
        case ThemingArgs_e.StyleChanged:
            if (this.Theming == Theming_e.AllowBorderOnly ||
                this.Theming == Theming_e.Disallow)
                break;

            styledark = thememanager.StyleDark;
            stylelight = thememanager.StyleLight;
            style = thememanager.Style;
            this.ChangeStyle();
            break;

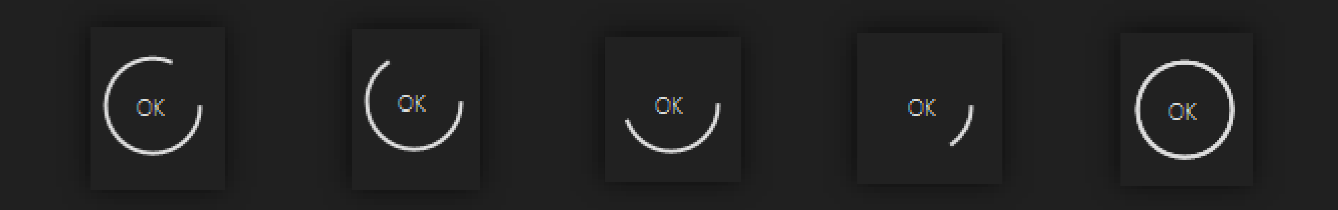
        case ThemingArgs_e.BorderSizeChanged:
            break;
```

```
                case ThemingArgs_e.BorderColorChanged:
                    break;

                default:
                    break;
            }
        }
        #endregion


        #region Custom Events
        ///
        public delegate void ThemingEventHandler(object send, ThemingEventArgs e);
        ///
        public event ThemingEventHandler ThemeChanged;
        ///
        protected virtual void OnThemeChanged(ThemingEventArgs e)
        {
            if (ThemeChanged != null)
                ThemeChanged.Invoke(this, e);
        }
        #endregion
    }
}
```

أنشئ نسخة من هذه الأداة ومرر مؤشر الفأرة فوقها ولاحظ:



يمكنك تطوير الأداة بجعلها أداة إظهار التقدم Progress.

الأدوات المعدلة بالرسم OnPaint

إذا تأملت الأدوات المتوفرة هنا أو هناك فإنك ستلاحظ أنها مستطيلة الشكل، وإذا علمت أن الأداة يُعاد رسمها في كل مرة يحدث عليها تغيير في منطقة الرسم، فإن ما يفترض أن يخطر على بالك هو إعادة تعريف الحدث Paint ورسم الأداة من الصفر بالشكل الذي تريد.

حدث الرسم Paint يحدث كلما تم إعادة رسم الأداة؛ لذلك عندما تنشئ أداة مخصصة جديدة أو تنشئ أداة ترث من أداة أخرى وترغب بتغيير مظهرها عليك إعادة تعريف الحدث OnPaint. [1]

لإعادة تعريف Overriding حدث ما، استخدم الكلمة override وبعدها اسم الحدث مسبوقًا بـ On، أي بالنسبة للحدث Paint يتم إعادة تعريفه بكتابة override OnPaint. ثم استدعِ الحدث OnPaint للفئة الأساس بكتابة base.OnPaint(e)، حيث e هي كائن من الفئة PaintEventArgs، تحوي بيانات عن ما سيتم رسمه، على شكل خاصيتين: ClipRectangle المستطيل الذي سيتم رسمه، وGraphics كائن الرسومات، والذي يعطيك إمكانيات الرسم المختلفة التي تطرقنا لها في الفصل الخامس. [2]

عند تأليف أداة مشتقة من الفئة Control، عليك تضمين الأكواد التي تسمح للأداة برسم نفسها وإعطاءها الشكل والمظهر المطلوبين. أما إذا كانت أداتك مشتقة من الفئة UserControl أو أي أداة رسومية أخرى، عليك – إن رغبت في تغيير مظهر الأداة – إعادة تعريف الطريقة OnPaint كما ذكرنا قبل أسطر. [3]

إذا كنت بين خيارين متاحين: إنشاء الأداة من أدوات أخرى أو رسمها، فلا تتردد باختيار الخيار الثاني، حتى لو كان الخيار الأصعب، إلا أنه الأفضل والأضمن والأكثر إتقانًا. إذ إنه يمكنك رسم أي شكل تريده، لديك الكائن Graphics، وتقنيات الرسم المختلفة التي يقدمها لك، وإبداعك.

---

[1] انظر المقال: الحدث Control.Paint
https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.control.paint?view=netcore-3.1

[2] انظلا المقال: إعادة تعريف الحدث OnPaint
https://docs.microsoft.com/en-us/dotnet/desktop/winforms/controls/overriding-the-onpaint-method?view=netframeworkdesktop-4.8

[3] راجع المقال: رسم الأدوات الخاصة
https://docs.microsoft.com/en-us/dotnet/desktop/winforms/controls/custom-control-painting-and-rendering?view=netframeworkdesktop-4.8

ButtonCircular



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class ButtonCircular : Eng27ButtonBase, IEng27Radial, IEng27Border
    {
        #region Constructors
        ///
        public ButtonCircular()
        {
            this.Height = 50;
        }
        #endregion


        #region Properties
        public Style_e Style ...
        public StyleDark StyleDark ...
        public StyleLight StyleLight ...
        public ThemeManager ThemeManager ...
        public Theming_e Theming ...

        private int radius = 25;
        ///
        public int Radius
        {
            get { return radius; }
            set { radius = value; this.Height = 2 * radius; }
        }

        private int bordersize = 1;
        ///
        public int BorderSize
        {
            get { return bordersize; }
            set { bordersize = value; this.Invalidate(); }
        }

        private Color bordercolor = Color.FromArgb(200, Color.Black);
        ///
        public Color BorderColor
        {
            get { return bordercolor; }
            set { bordercolor = value; this.Invalidate(); }
        }
```

```
        [Browsable(false)]
        [EditorBrowsable(EditorBrowsableState.Never)]
        public Size Size { get; set; }
        #endregion


        #region Methods
        void ChangeStyle()
        {
            switch (style)
            {
                case Style_e.Dark:
                    this.BackColor = styledark.BackColor;
                    this.ForeColor = styledark.ForeColor;
                    this.Font = styledark.Font;
                    break;

                case Style_e.Light:
                    this.BackColor = stylelight.ForeColor;
                    this.ForeColor = stylelight.BackColor;
                    this.Font = stylelight.Font;
                    break;

                default:
                    break;
            }
            if (thememanager != null)
            {
                if (thememanager.UserBorderSettings)
                    this.BorderColor = thememanager.BorderColor;
                else
                    this.BorderColor = this.ForeColor;
                this.BorderSize = thememanager.BorderSize;
            }
            else
                this.BorderColor = this.ForeColor;
        }
        #endregion


        #region Events
        protected override void OnPaint(PaintEventArgs pevent)
        {
            base.OnPaint(pevent);
            pevent.Graphics.SmoothingMode = SmoothingMode.AntiAlias;
            GraphicsPath gp = new GraphicsPath();

            int margin = bordersize * 3;
            gp.AddEllipse(
                margin,
                margin,
                radius * 2 - 2 * margin,
                radius * 2 - 2 * margin);

            this.Region = new System.Drawing.Region(gp);

            margin = bordersize * 4;
            pevent.Graphics.DrawEllipse(
                new Pen(bordercolor,
                    bordersize),
```

```
                new Rectangle(margin,
                    margin,
                    radius * 2 - 2 * margin,
                    radius * 2 - 2 * margin)
                    );
        }

        protected override void OnClientSizeChanged(EventArgs e)
        {
            radius = this.Width / 2;
        }

        protected override void SetBoundsCore
            (int x, int y, int width, int height, BoundsSpecified specified)
        {
            width = height;
            base.SetBoundsCore(x, y, width, height, specified);
        }

        void StyleLight_StyleChanged(object sender, System.EventArgs e) ...
        void StyleDark_StyleChanged(object sender, System.EventArgs e) ...

        private void thememanager_ThemeChanged(object sender, ThemingEventArgs e)
        {
            switch (e.Theming)
            {
                case ThemingArgs_e.StyleChanged:
                    if (this.Theming == Theming_e.AllowBorderOnly ||
                        this.Theming == Theming_e.Disallow)
                        break;

                    styledark = thememanager.StyleDark;
                    stylelight = thememanager.StyleLight;
                    style = thememanager.Style;
                    ChangeStyle();
                    break;

                case ThemingArgs_e.BorderSizeChanged:
                    if (this.Theming == Theming_e.AllowStyleOnly ||
                        this.Theming == Theming_e.Disallow)
                        break;
                    bordersize = thememanager.BorderSize;
                    break;

                case ThemingArgs_e.BorderColorChanged:
                    if (this.Theming == Theming_e.AllowStyleOnly ||
                        this.Theming == Theming_e.Disallow)
                        break;

                    bordercolor = thememanager.BorderColor;
                    break;

                default:
                    break;
            }
        }
        #endregion
```

```
        #region Custom Events
        ///
        public delegate void ThemingEventHandler(object send, ThemingEventArgs e);
        ///
        public event ThemingEventHandler ThemeChanged;
        ///
        protected virtual void OnThemeChanged(ThemingEventArgs e)
        {
            if (ThemeChanged != null)
                ThemeChanged.Invoke(this, e);
        }
        #endregion
    }
}
```

طوّر الأداة بإضافة معدد لخصائص الأداة تعطي المبرمج إمكانية اختيار شكل الزر، دائري أو مثلث أو مربع أو غيره من الأشكال.

CheckBoxCircular



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class CheckBoxCircular
        : CheckBox, IEng27Control, IEng27Border, IEng27Radial {
        #region Constructors
        ///
        public CheckBoxCircular() {
            this.SetStyle(
                ControlStyles.UserPaint | ControlStyles.AllPaintingInWmPaint,
                true);
            this.Padding = new Padding(5);
            this.AutoSize = false;
            this.Cursor = Cursors.Hand;
        }
        #endregion


        #region Properties
        private int radius = 25;
        ///
        public int Radius {
            get { return radius; }
            set { radius = value; this.Height = 2 * radius; }
        }
```

```
        private int bordersize = 2;
        ///
        public int BorderSize
        {
            get { return bordersize; }
            set { bordersize = value; this.Invalidate(); }
        }

        private Color bordercolor = Color.FromArgb(200, Color.Black);
        ///
        public Color BorderColor
        {
            get { return bordercolor; }
            set { bordercolor = value; this.Invalidate(); }
        }

        private Color focus_color = Color.FromArgb(61, 201, 129);
        ///
        public Color FocusColor
        {
            get { return focus_color; }
            set { focus_color = value; }
        }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public ThemeManager ThemeManager { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Theming_e Theming { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleDark StyleDark { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleLight StyleLight { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Style_e Style { get; set; }
        #endregion

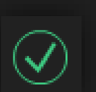

        #region Methods
        protected override void SetBoundsCore
            (int x, int y, int width, int height, BoundsSpecified specified)
        {
            width = height;
            base.SetBoundsCore(x, y, width, height, specified);
        }
        #endregion
```

```
        #region Events
        protected override void OnPaint(PaintEventArgs e)
        {
            this.OnPaintBackground(e);
            e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;
            using (var path = new GraphicsPath())
            {
                int w = this.Width, h = this.Height;
                path.AddLine(
                    new Point(w / 4, h / 2),
                    new Point(5 * w / 11, 5 * h / 7));
                path.AddLine(
                    new Point(5 * w / 11, 5 * h / 7),
                    new Point(3 * w / 4, h / 4));
                Color bordercolor = Checked ? this.BackColor : focus_color;
                Color fillcolor = Checked ? focus_color : this.BackColor;
                e.Graphics.FillEllipse(
                    new SolidBrush(fillcolor), new Rectangle(0, 0, w - 1, h - 1));
                e.Graphics.DrawPath(
                    new Pen(bordercolor, 2), path);
                e.Graphics.DrawEllipse(
                    new Pen(bordercolor, 1), new Rectangle(0, 0, w - 1, h - 1));
            }
        }

        protected override void OnClientSizeChanged(EventArgs e)
        {
            radius = this.Width / 2;
        }
        #endregion
    }
}
```

لاحظ:





PictureBoxCircular



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class PictureBoxCircular
        : PictureBox, IEng27Control, IEng27Border, IEng27Radial
    {
        #region Cosntructors
        ///
        public PictureBoxCircular()
        {

        }
        #endregion


        #region Properties
        private int radius = 25;
        ///
        public int Radius
        {
            get { return radius; }
            set { radius = value; this.Height = 2 * radius; }
        }

        private int bordersize = 2;
        ///
        public int BorderSize
        {
            get { return bordersize; }
            set { bordersize = value; this.Invalidate(); }
        }

        private Color bordercolor = Color.FromArgb(200, Color.Black);
        ///
        public Color BorderColor
        {
            get { return bordercolor; }
            set { bordercolor = value; this.Invalidate(); }
        }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public ThemeManager ThemeManager { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Theming_e Theming { get; set; }
```

```
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleDark StyleDark { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleLight StyleLight { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Style_e Style { get; set; }
        #endregion


        #region Methods
        protected override void SetBoundsCore
            (int x, int y, int width, int height, BoundsSpecified specified)
        {
            width = height;
            base.SetBoundsCore(x, y, width, height, specified);
        }
        #endregion


        #region Events
        protected override void OnPaint(PaintEventArgs pe)
        {
            base.OnPaint(pe);
            using (GraphicsPath gp = new GraphicsPath())
            {
                pe.Graphics.SmoothingMode = SmoothingMode.AntiAlias;

                int margin = 2 * bordersize;

                gp.AddEllipse(margin,
                    margin,
                    2 * radius - 2* margin,
                    2 * radius - 2 *margin);

                this.Region = new Region(gp);
                SolidBrush b = new SolidBrush(bordercolor);
                Pen pen = new Pen(b, bordersize);
                pe.Graphics.DrawEllipse(
                    pen,
                    margin,
                    margin,
                    2 * radius - 2 * margin,
                    2 * radius - 2 * margin);
            }
        }

        protected override void OnClientSizeChanged(EventArgs e)
        {
            radius = this.Width / 2;
        }
        #endregion
    }
}
```



يمكنك جعل الأداة إهليلجية الشكل (قطع ناقص) بتزويدها بنصفي أقطار (أو قطرين)، هما – أي القطرين – طول الأداة وعرضها. ويمكنك إضافة معدّد يقوم بتحديد الشكل المطلوب رسمه: دائري أو قطع ناقص أو غيره.



ButtonImage



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class ButtonImage : Eng27Control, IEng27Animation
    {
        #region Local Variables
        PictureBox picture = new PictureBox();
        Timer timer1 = new Timer();
        Timer timer2 = new Timer();
        Size picturesize;
        #endregion


        #region Constructors
        ///
        public ButtonImage()
        {
            this.Width = 60;
            this.Height = 60;

            this.InisializePictureSize();
            this.Controls.Add(picture);

            picture.SizeMode = PictureBoxSizeMode.StretchImage;
            this.image = picture.Image = Properties.Resources.zoom_image;
```

```
        timer1.Tick += timer1_Tick;
        timer2.Tick += timer2_Tick;
        timer1.Interval = 25; timer2.Interval = 25;

        picture.MouseEnter += picture_MouseEnter;
        picture.MouseLeave += picture_MouseLeave;

        picture.Click += picture_Click;

        picture.MouseDown += picture_MouseDown;
        picture.MouseUp += picture_MouseUp;
    }
    #endregion


    #region Properties
    private int indent = 5;
    ///
    public int Indent
    {
        get { return indent; }
        set
        {
            if (indent < maxindent)
                indent = value;
            this.InisializePictureSize();
        }
    }

    private int maxindent = 10;
    ///
    public int MaxIndent
    {
        get { return maxindent; }
        set
        {
            if (maxindent > indent)
                maxindent = value;
            this.InisializePictureSize();
            // else throw exception!
        }
    }

    private bool isanimating = false;
    ///
    public bool IsAnimating
    {
        get { return isanimating; }
    }

    private int steps = 4;
    ///
    public int Steps
    {
        get { return steps; }
        set { if (steps > 1) steps = value; }
    }
```

```
        int interval = 25;
        ///
        public int Interval
        {
            get { return interval; }
            set
            {
                interval = value;
                timer1.Interval = value;
                timer1.Interval = value;
            }
        }

        private Image image;
        ///
        public Image Image
        {
            get { return image; }
            set { image = value; picture.Image = value; }
        }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public ThemeManager ThemeManager { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Theming_e Theming { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleDark StyleDark { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleLight StyleLight { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Style_e Style { get; set; }
        #endregion


        #region Methods
        private void InisializePictureSize()
        {
            picturesize = picture.Size =
                new System.Drawing.Size(
                    this.Width - 2 * maxindent,
                    this.Height - 2 * maxindent);
            picture.Location = new Point(maxindent, maxindent);
        }

        protected override void SetBoundsCore
            (int x, int y, int width, int height, BoundsSpecified specified)
        {
            width = height;
            base.SetBoundsCore(x, y, width, height, specified);
        }
        #endregion
```

```
        #region Events
        protected override void OnSizeChanged(EventArgs e)
        {
            this.InisializePictureSize();
            base.OnSizeChanged(e);
        }

        void picture_MouseUp(object sender, MouseEventArgs e)
        {
            picture.Width += steps;
            picture.Height += steps;
            picture.Left -= (int)(steps / 2);
            picture.Top -= (int)(steps / 2);
            this.OnMouseUp(e);
        }

        void picture_MouseDown(object sender, MouseEventArgs e)
        {
            picture.Width -= steps;
            picture.Height -= steps;
            picture.Left += (int)(steps / 2);
            picture.Top += (int)(steps / 2);
            this.OnMouseDown(e);
        }

        void picture_Click(object sender, EventArgs e)
        {
            this.OnClick(e);
        }

        void timer2_Tick(object sender, EventArgs e)
        {
            if (picture.Width > picturesize.Width)
            {
                picture.Width -= steps;
                picture.Height -= steps;
                picture.Left += (int)(steps / 2);
                picture.Top += (int)(steps / 2);
            }

            else
            {

                picture.Height = picture.Width = picturesize.Width;
                picture.Left = maxindent;
                picture.Top = maxindent;
                timer2.Enabled = false;
            }
        }

        void timer1_Tick(object sender, EventArgs e)
        {
            if (picture.Top > indent)
            {
                picture.Width += steps;
                picture.Height += steps;
                picture.Left -= (int)(steps / 2);
                picture.Top -= (int)(steps / 2);
            }
```

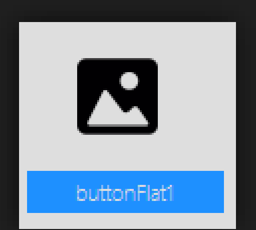
```
            else
            {
                picture.Height = picture.Width = (int)(this.Width - 2 * indent);
                picture.Left = indent;
                picture.Top = indent;
                timer1.Enabled = false;
            }
        }

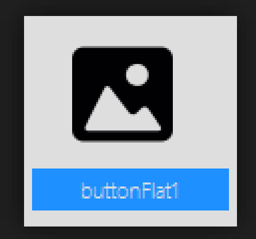
        void picture_MouseLeave(object sender, EventArgs e)
        {
            timer1.Enabled = false;
            timer2.Enabled = true;
        }

        private void picture_MouseEnter(object sender, EventArgs e)
        {
            timer1.Enabled = true;
        }
        #endregion
    }
}
```

ستحتاج لإضافة صورة باسم zoom_Image لمصادر المشروع.

لاحظ الأداة قبل دخول مؤشر الفأرة عليها وبعد دخوله (وضعت الزر الثاني لتستطيع مقارنة الحجم):





يمكنك تطوير الأداة بإعطاء المبرمج الخيار لجعل تغيير الحجم للخارج أو الداخل.

ButtonGradiant



```
using Eng27.Components;
using Eng27.Enums;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class ButtonGradiant : Eng27ButtonBase
    {
        #region Constructors
        ///
        public ButtonGradiant()
        {
            gradiantcolors = new GradiantColors();
            this.FlatStyle = FlatStyle.Standard;
        }
        #endregion


        #region Properties
        private GradiantColors gradiantcolors;
        ///
        [Browsable(true)]
        [TypeConverter(typeof(ExpandableObjectConverter))]
        [RefreshProperties(RefreshProperties.Repaint)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Content)]
        public GradiantColors GradiantColors
        {
            get { return gradiantcolors; }
            set { gradiantcolors = value; this.Invalidate(); }
        }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public ThemeManager ThemeManager { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Theming_e Theming { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleDark StyleDark { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleLight StyleLight { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Style_e Style { get; set; }
        #endregion
```

```
        #region Events
        protected override void OnPaint(PaintEventArgs pe)
        {
            // Calling the base class OnPaint
            base.OnPaint(pe);
            // Create two semi-transparent colors
            Color c1 = Color.FromArgb(
                gradiantcolors.Transparent1, gradiantcolors.Color1);
            Color c2 = Color.FromArgb(
                gradiantcolors.Transparent2, gradiantcolors.Color2);
            Brush b = new LinearGradientBrush(this.ClientRectangle,
                c1, c2, 10);
            pe.Graphics.FillRectangle(b, this.ClientRectangle);
            b.Dispose();
        }
        #endregion
    }
}
```

ألوان هذه الأداة يحددها كائن من الفئة GradiantColors، الموجودة في مجال الأسماء Eng27 مباشرة:



```
using System.Drawing;

namespace Eng27
{
    ///
    public class GradiantColors
    {
        ///
        public GradiantColors()
        {
            color1 = Color.DarkBlue;
            color2 = Color.LightGreen;
            transparent1 = transparent2 = 64;
        }

        private int transparent1;
        ///
        public int Transparent1
        {
            get { return transparent1; }
            set { transparent1 = value; }
        }

        private int transparent2;
        ///
        public int Transparent2
        {
            get { return transparent2; }
            set { transparent2 = value; }
        }
```

```
        private Color color1;
        ///
        public Color Color1
        {
            get { return color1; }
            set { color1 = value; }
        }

        private Color color2;
        ///
        public Color Color2
        {
            get { return color2; }
            set { color2 = value; }
        }
    }
}
```

يمكنك تطوير الأداة بجعلها تقبل ألوانًا أكثر، أو أن يكون التلوين متدرجًا بشكل غير خطي، أو حتى بإعطاء المبرمج إمكانية اختيار زاوية ميل اللون المتدرج.

PanelGradiant



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class PanelGradiant : Panel, IEng27Control
    {
        #region Constructors
        ///
        public PanelGradiant()
        {
            gradiantcolors = new GradiantColors();
        }
        #endregion
```

```
        #region Properties
        private GradiantColors gradiantcolors;
        ///
        public GradiantColors GradiantColors
        {
            get { return gradiantcolors; }
            set { gradiantcolors = value; }
        }

        public ThemeManager ThemeManager ...
        public Style_e Style ...
        public StyleDark StyleDark ...
        public StyleLight StyleLight ...
        public Theming_e Theming ...
        #endregion


        #region Methods
        void ChangeStyle() ...
        #endregion


        #region Events
        protected override void OnPaint(PaintEventArgs pe)
        {
            base.OnPaint(pe);

            Color c1 = Color.FromArgb
                (gradiantcolors.Transparent1, gradiantcolors.Color1);
            Color c2 = Color.FromArgb
                (gradiantcolors.Transparent2, gradiantcolors.Color2);
            Brush b = new LinearGradientBrush(this.ClientRectangle,
                c1, c2, 10);
            pe.Graphics.FillRectangle(b, this.ClientRectangle);
            b.Dispose();
        }

        void StyleLight_StyleChanged(object sender, System.EventArgs e) ...
        void StyleDark_StyleChanged(object sender, System.EventArgs e) ...
        void thememanager_ThemeChanged(object sender, ThemingEventArgs e) ...
        #endregion
    }
}
```

تعتمد هذه الأداة على الفئة GradiantColors المستخدمة في الفقرة السابقة.



PanelRounded



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class PanelRounded : Panel, IEng27Control, IEng27Radial
    {
        #region Constructors
        ///
        public PanelRounded() { }
        #endregion


        #region Properties
        private int radius = 25;
        ///
        public int Radius
        {
            get { return radius; }
            set { radius = value; this.Invalidate(); }
        }

        public ThemeManager ThemeManager ...
        public Style_e Style ...
        public StyleDark StyleDark ...
        public StyleLight StyleLight ...
        public Theming_e Theming ...
        #endregion


        #region Methods
        ///
        public static GraphicsPath RoundedRect(Rectangle bounds, int radius)
        {
            int diameter = radius * 2;
            Size size = new Size(diameter, diameter);
            Rectangle arc = new Rectangle(bounds.Location, size);
            GraphicsPath path = new GraphicsPath();

            if (radius == 0) {
                path.AddRectangle(bounds);
                return path;
            }

            // top left arc
            path.AddArc(arc, 180, 90);

            // top right arc
            arc.X = bounds.Right - diameter;
            path.AddArc(arc, 270, 90);
```

```
            // bottom right arc
            arc.Y = bounds.Bottom - diameter;
            path.AddArc(arc, 0, 90);

            // bottom left arc
            arc.X = bounds.Left;
            path.AddArc(arc, 90, 90);

            path.CloseFigure();
            return path;
        }

        void ChangeStyle() ...
        #endregion


        #region Events
        protected override void OnPaint(PaintEventArgs e)
        {
            GraphicsPath shape = new GraphicsPath();

            shape = RoundedRect(new Rectangle(0, 0, Width, Height), radius);

            this.Region = new System.Drawing.Region(shape);
        }

        void StyleLight_StyleChanged(object sender, System.EventArgs e) ...
        void StyleDark_StyleChanged(object sender, System.EventArgs e) ...
        void thememanager_ThemeChanged(object sender, ThemingEventArgs e) ...
        #endregion
    }
}
```

PanelShadow



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Runtime.InteropServices;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class PanelShadow : Panel, IEng27Control
    {
        #region DllImports
        [DllImport("user32.dll")]
        static extern IntPtr GetWindowDC(IntPtr hWnd);
        [DllImport("User32.dll")]
        static extern int ReleaseDC(IntPtr hWnd, IntPtr hDC);
        #endregion


        #region Constructors
        ///
        public PanelShadow()
        {
            this.BorderStyle = BorderStyle.Fixed3D;
            this.Paint += ParentPaint;
        }
        #endregion


        #region Properties
        private Color shadowcolor = Color.Black;
        ///
        public Color ShadowColor
        {
            get { return shadowcolor; }
            set { shadowcolor = value; }
        }

        private int shadow_transparency = 128;
        ///
        public int ShadowTransparency
        {
            get { return shadow_transparency; }
            set { shadow_transparency = value; }
        }

        public ThemeManager ThemeManager ...
        public Style_e Style ...
        public StyleDark StyleDark ...
        public StyleLight StyleLight ...
        public Theming_e Theming ...
        #endregion
```

```
        #region Methods
        protected override void WndProc(ref Message m)
        {
            const int WM_NCPAINT = 133;
            if (m.Msg == WM_NCPAINT)
            {
                IntPtr hdc = GetWindowDC(m.HWnd);
                Graphics g = Graphics.FromHdc(hdc);
                Rectangle rDraw =
                    new Rectangle(0, 0, this.Width - 1, this.Height - 1);
                Pen pBottom = new Pen(Color.Gray, 3);
                Pen pTop = new Pen(Color.White, 3);
                g.DrawRectangle(pBottom, rDraw);
                Point[] pts = new Point[3];

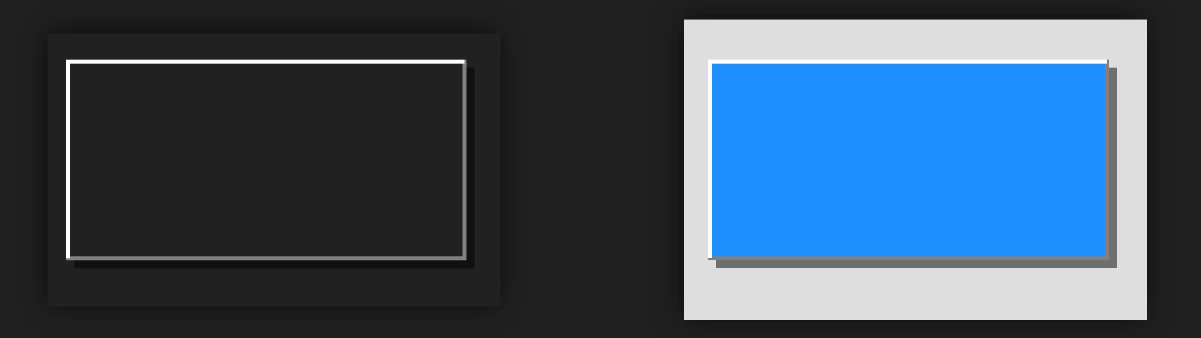
                pts[0] = new Point(0, this.Height - 1);
                pts[1] = new Point(0, 0);
                pts[2] = new Point(this.Width - 1, 0);


                g.DrawLines(pTop, pts);
                ReleaseDC(this.Handle, hdc);
            }
            else
            {
                base.WndProc(ref m);
            }
        }

        void ChangeStyle() ...
        #endregion


        #region Events
        private void ParentPaint(object sender, PaintEventArgs e)
        {
            Graphics g = this.Parent.CreateGraphics();
            Matrix mx = new Matrix(1F, 0, 0, 1F, 4, 4);
            Rectangle rdraw =
                new Rectangle(this.Left, this.Top, this.Width, this.Height);
            g.Transform = mx;
            g.FillRectangle(
                new SolidBrush(
                    Color.FromArgb(shadow_transparency, shadowcolor)), rdraw);
            g.Dispose();
        }

        void StyleLight_StyleChanged(object sender, System.EventArgs e) ...
        void StyleDark_StyleChanged(object sender, System.EventArgs e)  ...
        void thememanager_ThemeChanged(object sender, ThemingEventArgs e) ...
        #endregion
    }
}
```



هناك طريقة أخرى لرسم الظل، بالاعتماد على مجموعة من الصور، راجع هذا المقال [1].

ToggleSwitch



```
using Eng27.Components;
using Eng27.Enums;
using Eng27.Interfaces;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class ToggleSwitch : CheckBox, IEng27Control
    {
        #region Constructors
        ///
        public ToggleSwitch() {
            this.SetStyle(
                ControlStyles.UserPaint | ControlStyles.AllPaintingInWmPaint,
                true);
            this.Padding = new Padding(6);
            togglecolors = new ToggleSwitchColors();
            this.AutoSize = false;
        }
        #endregion


        #region Properties
        private ToggleSwitchColors togglecolors;
        ///
        [Browsable(true)]
        [TypeConverter(typeof(ExpandableObjectConverter))]
        [RefreshProperties(RefreshProperties.Repaint)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Content)]
```

[1] مقال من موقع CodeProject
https://www.codeproject.com/Articles/19243/Transparent-drop-shadow-in-C-GDI-Windows-Forms

```
        public ToggleSwitchColors ToggleColors
        {
            get { return togglecolors; }
            set { togglecolors = value; }
        }

        public ThemeManager ThemeManager ...
        public Style_e Style ...
        public StyleDark StyleDark ...
        public StyleLight StyleLight ...
        public Theming_e Theming ...
        #endregion


        #region Methods
        void ChangeStyle() ...
        #endregion


        #region Events
        protected override void OnPaint(PaintEventArgs e)
        {
            this.OnPaintBackground(e);
            e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;
            using (var path = new GraphicsPath())
            {
                var d = Padding.All;
                var r = this.Height - 2 * d;
                path.AddArc(d, d, r, r, 90, 180);
                path.AddArc(this.Width - r - d, d, r, r, -90, 180);
                path.CloseFigure();
                Brush fillpath = Checked ?
                    togglecolors.ControlChecked : togglecolors.ControlUnChecked;
                e.Graphics.FillPath(fillpath, path);
                r = Height - 1;
                var rect = Checked ? new Rectangle(Width - r - 1, 0, r, r)
                                   : new Rectangle(0, 0, r, r);
                Brush fillellipse;
                if (CheckState == System.Windows.Forms.CheckState.Checked)
                    fillellipse = togglecolors.HandleChecked;
                else if (CheckState == CheckState.Indeterminate)
                    fillellipse = togglecolors.HandleIndeterminate;
                else
                    fillellipse = togglecolors.HandleUnChecked;

                e.Graphics.FillEllipse(fillellipse, rect);
            }
        }

        void StyleLight_StyleChanged(object sender, System.EventArgs e) ...
        void StyleDark_StyleChanged(object sender, System.EventArgs e) ...
        void thememanager_ThemeChanged(object sender, ThemingEventArgs e) ...
        #endregion
    }
}
```

أما الفئة ToggleSwitchColors:



```
using System.Drawing;

namespace Eng27 {
    ///
    public class ToggleSwitchColors {
        ///
        public ToggleSwitchColors() {
            controlChecked = Brushes.DarkGray;
            controlUnChecked = Brushes.LightGray;
            handleChecked = Brushes.Green;
            handleUnChecked = Brushes.WhiteSmoke;
            handleIndeterminate = Brushes.LightGreen;
        }

        private Brush controlUnChecked;
        ///
        public Brush ControlUnChecked
        {
            get { return controlUnChecked; }
            set { controlUnChecked = value; }
        }

        private Brush controlChecked;
        ///
        public Brush ControlChecked
        {
            get { return controlChecked; }
            set { controlChecked = value; }
        }

        private Brush handleIndeterminate;
        ///
        public Brush HandleIndeterminate
        {
            get { return handleIndeterminate; }
            set { handleIndeterminate = value; }
        }

        private Brush handleUnChecked;
        ///
        public Brush HandleUnChecked
        {
            get { return handleUnChecked; }
            set { handleUnChecked = value; }
        }

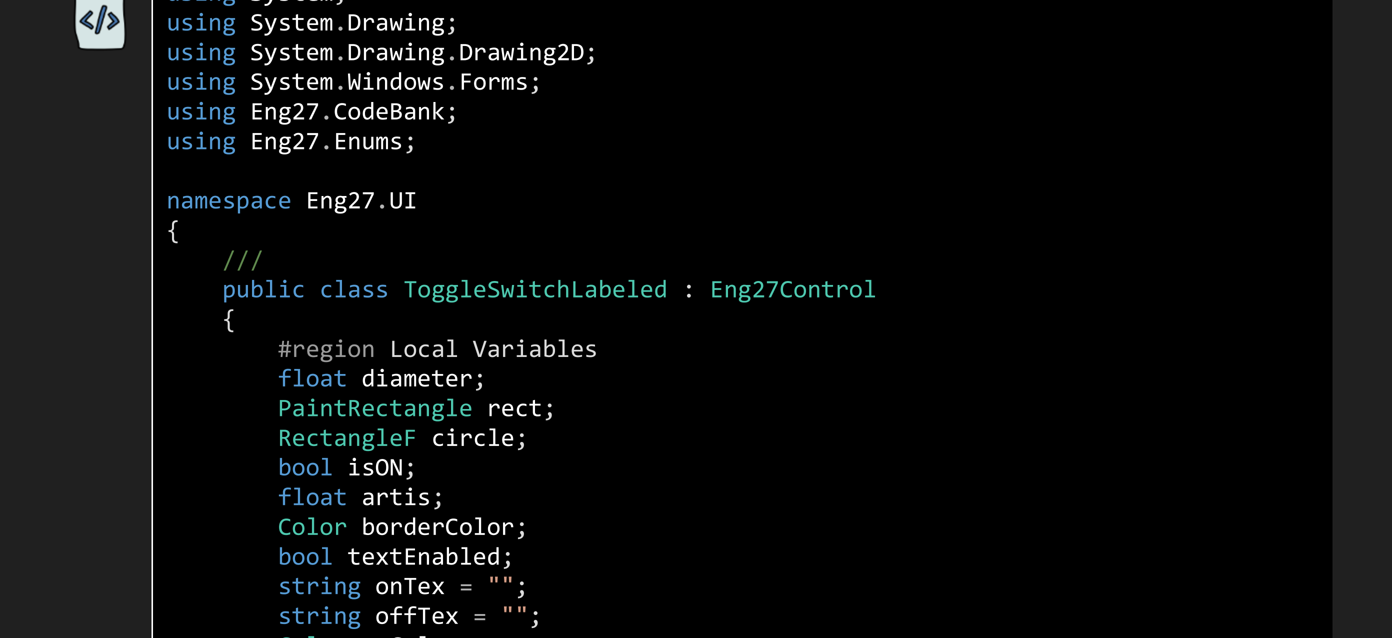
        private Brush handleChecked;
        ///
        public Brush HandleChecked
        {
            get { return handleChecked; }
            set { handleChecked = value; }
        }
    }
}
```



لست مضطرًا لتسمية الخصائص بنفس اسم الفئات التي تمثلها، لكنه يفضل ذلك إذا كانت الخصائص تمثّل كائنات سيتعامل معها المبرمج، مثل الفئة ThemeManager.



ToggleSwitchLabeled



```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;
using Eng27.CodeBank;
using Eng27.Enums;

namespace Eng27.UI
{
    ///
    public class ToggleSwitchLabeled : Eng27Control
    {
        #region Local Variables
        float diameter;
        PaintRectangle rect;
        RectangleF circle;
        bool isON;
        float artis;
        Color borderColor;
        bool textEnabled;
        string onTex = "";
        string offTex = "";
        Color onCol;
        Color offCol;
        Timer painTicker = new Timer();
        #endregion
```

```
        #region Constructors
        ///
        public ToggleSwitchLabeled()
        {
            this.Cursor = Cursors.Hand;
            this.DoubleBuffered = true;
            artis = 4f;
            diameter = 30f;
            textEnabled = true;

            rect = new PaintRectangle(
                2f * diameter,
                diameter + 2f,
                diameter / 2f,
                1f,
                1f);

            circle = new RectangleF(
                1f,
                1f,
                diameter,
                diameter);

            isON = false;
            borderColor = Color.LightGray;
            painTicker.Tick += new EventHandler(this.paintTicker_Tick);
            painTicker.Interval = 1;
            onCol = Color.FromArgb(94, 148, 255);
            offCol = Color.DarkGray;
            this.ForeColor = Color.White;
            onTex = "ON";
            offTex = "OFF";
        }
        #endregion


        #region Properties
        ///
        public bool TextEnabled
        {
            get { return this.textEnabled; }
            set
            {
                textEnabled = value;
                base.Invalidate();
            }
        }

        ///
        public bool IsOn
        {
            get { return this.isON; }
            set
            {
                painTicker.Stop();
                isON = value;
                painTicker.Start();
                if (this.ToggleValueChanged != null)
                {
                    this.ToggleValueChanged(this, EventArgs.Empty);
```

```
            }
        }
    }

    ///
    public Color BorderColor
    {
        get { return this.borderColor; }
        set
        {
            borderColor = value;
            base.Invalidate();
        }
    }

    ///
    public string OnText
    {
        get { return onTex; }
        set
        {
            onTex = value;
            base.Invalidate();
        }
    }

    ///
    public string OffText
    {
        get { return offTex; }
        set
        {
            offTex = value;
            base.Invalidate();
        }
    }

    ///
    public Color OnColor
    {
        get { return onCol; }
        set
        {
            onCol = value;
            base.Invalidate();
        }
    }

    ///
    public Color OffColor
    {
        get { return offCol; }
        set
        {
            offCol = value;
            base.Invalidate();
        }
    }
    #endregion
```

```
        #region Methods
        public override void ChangeStyle()
        {
            switch (this.Style)
            {
                case Style_e.Dark:
                    this.BackColor = this.StyleDark.BackColor;
                    this.ForeColor = this.StyleDark.ForeColor;
                    this.Font = this.StyleDark.Font;
                    break;

                case Style_e.Light:
                    this.BackColor = this.StyleLight.ForeColor;
                    this.ForeColor = this.StyleLight.BackColor;
                    this.Font = this.StyleLight.Font;
                    break;

                default:
                    break;
            }
        }
        #endregion


        #region Events
        protected override void OnEnabledChanged(EventArgs e)
        {
            base.Invalidate();
            base.OnEnabledChanged(e);
        }

        protected override void OnMouseDown(MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
            {
                isON = !isON;
                this.IsOn = isON;
                base.OnMouseClick(e);
            }
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            e.Graphics.SmoothingMode = SmoothingMode.HighQuality;
            if (base.Enabled)
            {
                Pen pen;
                using (SolidBrush brush = new SolidBrush(isON ? onCol : offCol))
                {
                    e.Graphics.FillPath((Brush)brush, rect.Path);
                }
                using (pen = new Pen(borderColor, 2f))
                {
                    e.Graphics.DrawPath(pen, rect.Path);
                }
                if (this.textEnabled)
                {
                    using (Font font = new Font(
                        "Century Gothic",
                        (8.2f * diameter) / 30f,
```

```
                    (FontStyle)FontStyle.Bold))
                {
                    SolidBrush b = new SolidBrush(this.ForeColor);
                    int height = TextRenderer.MeasureText(onTex, font).Height;
                    float num2 = (diameter - height) / 2f;
                    e.Graphics.DrawString(onTex, font, b, 5f, num2 + 1f);
                    height = TextRenderer.MeasureText(offTex, font).Height;
                    num2 = (diameter - height) / 2f;
                    e.Graphics.DrawString
                        (offTex, font, b, diameter + 2f, num2 + 1f);
                }
                using (SolidBrush brush2 =
                    new SolidBrush("#FFFFFF".FromHex())) // راجع الإضافة بعد كود هذه الفئة
                {
                    e.Graphics.FillEllipse((Brush)brush2, circle);
                }
                using (pen = new Pen(Color.LightGray, 1.2f))
                {
                    e.Graphics.DrawEllipse(pen, circle);
                }
            }
            else
            {
                using (SolidBrush brush3 =
                    new SolidBrush("#FFFFFF".FromHex()))
                {
                    using (SolidBrush brush4 =
                        new SolidBrush("#FFFFFF".FromHex()))
                    {
                        e.Graphics.FillPath((Brush)brush3, rect.Path);
                        e.Graphics.FillEllipse((Brush)brush4, circle);
                        e.Graphics.DrawEllipse(Pens.DarkGray, circle);
                    }
                }
            }
        }
        base.OnPaint(e);
    }

    protected override void OnResize(EventArgs e)
    {
        base.Width = (base.Height - 2) * 2;
        diameter = base.Width / 2;
        artis = (4f * diameter) * 30f;
        rect = new PaintRectangle(
            2f * diameter, diameter + 2f, diameter / 2f, 1f, 1f);
        circle = new RectangleF(!isON ?
            1f : ((base.Width - diameter) - 1f), 1f, diameter, diameter);
        base.OnResize(e);
    }

    private void paintTicker_Tick(object sender, EventArgs e)
    {
        float x = circle.X;

        if (isON)
        {
            if ((x + artis) <= ((base.Width - diameter) - 1f))
            {
                x += artis;
```

```
                    circle = new RectangleF(x, 1f, diameter, diameter);
                    base.Invalidate();
                }

                else
                {
                    x = (base.Width - diameter) - 1f;
                    circle = new RectangleF(x, 1f, diameter, diameter);
                    base.Invalidate();
                    painTicker.Stop();
                }
            }

            else if ((x - artis) >= 1f)
            {
                x -= artis;
                circle = new RectangleF(x, 1f, diameter, diameter);
            }

            else
            {
                x = 1f;
                circle = new RectangleF(x, 1f, diameter, diameter);
                base.Invalidate();
                painTicker.Stop();
            }
        }
        #endregion


        #region Custom Events
        ///
        public delegate void ToggleChangedEventHandler(object sendr, EventArgs e);
        ///
        public event ToggleChangedEventHandler ToggleValueChanged;
        #endregion
    }
}
```

تعمدت استخدام أسلوب تصميم هذه الفئة بحيث يختلف عن الفئات السابقة، فقد أنشأت مفوضًا يأخذ كائنًا من الفئة EventArgs لتمثيل حيثيات الحدث، مع أن المفوض EventHandler الافتراضي يقوم بهذه المهمة وزيادة! ومرادي من هذا لفت انتباهك ليس إلا. كما أن ربط الحدث Tick الخاص بالمؤقت painTicker لم أنشئه كما في الفئات السابقة.

بين أكواد هذه الفئة تم استدعاء الطريقة FromHex، من أجل القيم النصية، والتي لن تجدها لديك، وإنما عليك إضافتها للفئة String، من خلال الإضافات Extensions. هذه الفئة موجودة في مجال الأسماء Eng27.CodeBank:



```
using System.Drawing;

namespace Eng27.CodeBank
{
    ///
    internal static class ExtenssionMethods
    {
        /// <summary>
        /// Converts a string to Color.
        /// </summary>
        /// <param name="hex">An HTML color.</param>
        /// <returns></returns>
        public static Color FromHex(this string hex)
        {
            return ColorTranslator.FromHtml(hex);
        }
    }
}
```

الفئة السابقة ستجعل المتغيرات النصية قادرة على القيام بالوظيفة FromHtml، أي أنك إذا كتبت نقطة أما اسم أي متغير نصي ستجد هذه الطريقة.

استخدم هذا الأسلوب لإضافة وظائف لأنواع البيانات المشهورة لإغنائها.

كما تم الاعتماد على الفئة PaintRectangle:



```
using System.Drawing;
using System.Drawing.Drawing2D;

namespace Eng27
{
    ///
    internal class PaintRectangle
    {
        private Point location;
        private float radius;
        private GraphicsPath grPath;
        private float x;
        private float y;
        private float width;
        private float height;
        RectangleF rect;

        ///
        public PaintRectangle() { }
```

```
        ///
        public PaintRectangle
            (float width, float height, float radius, float x = 0f, float y = 0f)
        {
            this.location = new Point(0, 0);
            this.radius = radius;
            this.x = x;
            this.y = y;
            this.height = height;
            this.width = width;
            this.grPath = new GraphicsPath();
            this.rect = new RectangleF(this.x, this.y, this.width, this.height);
            if (radius <= 0f)
            {
                this.grPath.AddRectangle(new RectangleF(x, y, width, height));
            }
            else
            {
                RectangleF ef = new RectangleF(x, y, 2f * radius, 2f * radius);
                RectangleF ef2 = new RectangleF
                    ((width - (2f * radius)) - 1f, x, 2f * radius, 2f * radius);
                RectangleF ef3 = new RectangleF
                    (x, (height - (2f * radius)) - 1f, 2f * radius, 2f * radius);
                RectangleF ef4 = new RectangleF((width - (2f * radius)) - 1f,
                    (height - (2f * radius)) - 1f, 2f * radius, 2f * radius);
                this.grPath.AddArc(ef, 180f, 90f);
                this.grPath.AddArc(ef2, 270f, 90f);
                this.grPath.AddArc(ef4, 0f, 90f);
                this.grPath.AddArc(ef3, 90f, 90f);
                this.grPath.CloseAllFigures();
            }
        }

        ///
        public GraphicsPath Path
        {
            get { return grPath; }
            set { grPath = value; }
        }

        ///
        public RectangleF Rect
        {
            get { return rect; }
            set { rect = value; }
        }

        ///
        public float Radius
        {
            get { return radius; }
            set { radius = value; }
        }
    }
}
```



ProgressBarCircular



```
using Eng27.Enums;
using Eng27.Interfaces;
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class ProgressBarCircular : Eng27Control, IEng27Animation, IEng27Border
    {
        #region Local Variables
        int value;
        int maxValue = 100;
        int minValue;
        int outerRadius;
        int innerRadius;
        int stroke;
        bool automaticFontCalculation;
        bool allowText;
        bool transparency;
        Color progressColor;
        SolidBrush brush;
        Pen pen;
        Timer timer = new Timer();
        int old_value;
        int current;
        #endregion


        #region Constructors
        ///
        public ProgressBarCircular() {
            value = this.maxValue;
            minValue = 0;
            progressColor = Color.Green;
            innerRadius = 30;
            outerRadius = 50;
            stroke = 10;
            this.MinimumSize = new Size(60, 60);
```

```
            automaticFontCalculation = true;
            allowText = true;
            transparency = true;
            brush = new SolidBrush(this.progressColor);
            pen = new Pen(bordercolor, bordersize);
            base.SetStyle(
                ControlStyles.OptimizedDoubleBuffer |
                ControlStyles.AllPaintingInWmPaint |
                ControlStyles.SupportsTransparentBackColor |
                ControlStyles.UserPaint,
                true);
            this.Location = new Point(100, 100);
            this.BackColor = Color.Transparent;
            timer.Tick += new EventHandler(timer_Tick);
            timer.Interval = interval;
            timer.Enabled = true;
        }
        #endregion


        #region Properties
        private Color bordercolor = Color.Black;
        ///
        public Color BorderColor
        {
            get { return bordercolor; }
            set { bordercolor = value; InitializePen(); }
        }

        private int bordersize = 1;
        ///
        public int BorderSize
        {
            get { return bordersize; }
            set { bordersize = value; InitializePen(); }
        }

        private bool showborder = true;
        ///
        public bool ShowBorder
        {
            get { return showborder; }
            set { showborder = value; }
        }

        int interval = 10;
        ///
        public int Interval
        {
            get { return interval; }
            set { interval = value; timer.Interval = value; }
        }

        private int steps = 5;
        ///
        public int Steps
        {
            get { return steps; }
            set { steps = value; }
        }
```

```
        private bool isanimating = false;
        ///
        public bool IsAnimating
        {
            get { return isanimating; }
        }

        ///
        public int Value
        {
            get { return current; }
            set
            {
                old_value = this.value;
                if (value >= minValue && value <= maxValue)
                {
                    current = value;
                    timer.Start();
                }
            }
        }

        ///
        public int MaxValue
        {
            get { return maxValue; }
            set
            {
                maxValue = value;
                if (this.value > value) this.value = value;
                base.Invalidate();
            }
        }

        ///
        public int MinValue
        {
            get { return minValue; }
            set
            {
                minValue = value;
                if (this.value < value) this.value = value;
                base.Invalidate();
            }
        }

        ///
        public Color ProgrssColor
        {
            get { return progressColor; }
            set
            {
                progressColor = value;
                brush = new SolidBrush(progressColor);
            }
        }
```

```
        ///
        public bool AutomaticFontCalculation
        {
            get { return automaticFontCalculation; }
            set
            {
                automaticFontCalculation = value;
                base.Invalidate();
            }
        }

        ///
        public int Stroke
        {
            get { return stroke; }
            set
            {
                if ((outerRadius - value) >= 15)
                {
                    stroke = value;
                }
                innerRadius = outerRadius - stroke;
                base.Invalidate();
            }
        }

        ///
        public bool AllowText
        {
            get { return allowText; }
            set
            {
                allowText = value;
                base.Invalidate();
            }
        }

        ///
        public bool Transparency
        {
            get { return transparency; }
            set
            {
                transparency = value;
                base.Invalidate();
            }
        }
        #endregion


        #region Methods
        void InitializePen()
        {
            pen = new Pen(bordercolor, bordersize);
        }
```

```
        public override void ChangeStyle()
        {
            switch (this.Style)
            {
                case Style_e.Dark:
                    this.BackColor = this.StyleDark.BackColor;
                    this.ForeColor = this.StyleDark.ForeColor;
                    this.Font = this.StyleDark.Font;
                    break;

                case Style_e.Light:
                    this.BackColor = this.StyleLight.ForeColor;
                    this.ForeColor = this.StyleLight.BackColor;
                    this.Font = this.StyleLight.Font;
                    break;

                default:
                    break;
            }
        }
        #endregion


        #region Events
        protected override void OnPaint(PaintEventArgs e)
        {
            if (this.transparency && (base.Parent != null))
            {
                Bitmap bitmap = new Bitmap(base.Parent.Width, base.Parent.Height);
                foreach (Control control in base.Parent.Controls)
                {
                    if (control.Bounds.IntersectsWith(base.Bounds) &
                        control != this)
                    {
                        control.DrawToBitmap(bitmap, control.Bounds);
                    }
                }
                e.Graphics.DrawImage((Image)bitmap, -base.Left, -base.Top);
            }
            float num = (value * 360f) / ((float)maxValue);
            e.Graphics.SmoothingMode = SmoothingMode.HighQuality;

            RectangleF ef = new RectangleF(
                0f, 0f, (float)(outerRadius * 2), (float)(outerRadius * 2));

            RectangleF ef1 = new RectangleF(
                (float)(outerRadius - innerRadius),
                (float)(outerRadius - innerRadius),
                (float)(this.innerRadius * 2),
                (float)(innerRadius * 2));

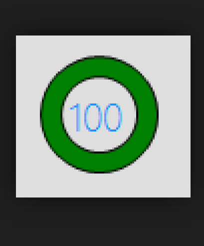
            using (GraphicsPath path = new GraphicsPath())
            {
                path.AddArc(ef, num - 90f, -num);
                if (allowText)
                {
                    path.AddArc(ef1, -90f, num);
                }
                else
                {
```

```
                path.AddLine(
                    new Point(outerRadius, 0),
                    new Point(outerRadius, outerRadius));
            }
            path.CloseFigure();
            brush = new SolidBrush(progressColor);
            e.Graphics.FillPath((Brush)brush, path);
            if (showborder) e.Graphics.DrawPath(pen, path);
            if (this.allowText)
            {
                string str =
                    (((this.value * 100.0) /
                    ((double)maxValue))).ToString("0");
                float num2 = 1f;
                if (this.automaticFontCalculation)
                {
                    string str2 = "100";
                    Size size2 = TextRenderer.MeasureText(str2, this.Font);
                    float num3 = innerRadius * 1.2f;
                    num2 = num3 / ((float)size2.Width);
                }
                Font font = new Font(this.Font.Name, this.Font.Height * num2);
                Size size = TextRenderer.MeasureText(str, font);
                float num4 = ((float)((2 * outerRadius) - size.Width)) / 2f;
                float num5 = ((float)((2 * outerRadius) - size.Height)) / 2f;
                using (brush = new SolidBrush(this.ForeColor))
                {
                    e.Graphics.DrawString(
                        str, font, (Brush)brush, num4 + 1f, num5);
                }
            }
        }
    }

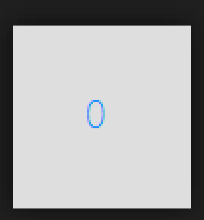
    protected override void OnResize(EventArgs e)
    {
        if (allowText && ((outerRadius - stroke) <= 15))
        {
            stroke--;
        }
        else
        {
            base.OnResize(e);
            base.Height = base.Width;
            outerRadius = (base.Width / 2) - 1;
            innerRadius = outerRadius - stroke;
            base.Invalidate();
        }
    }

    private void timer_Tick(object sender, EventArgs e)
    {
        for (int i = 0; i < 3; i++)
        {
            int num2 = current - old_value;
            int num3 = maxValue / 100;
            int num4 = (num3 < 1) ? 1 : num3;
            old_value +=
                (Math.Abs(num2) < 2) ? num2 : (num4 * Math.Sign(num2));
            value = old_value;
```

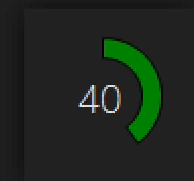
```
                base.Invalidate();
                if (current == old_value)
                {
                    timer.Stop();
                }
            }
        }
        #endregion
    }
}
```

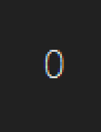












TextBoxRounded



```
using Eng27.CodeBank;
using Eng27.Enums;
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class TextBoxRounded : Eng27Control
    {
        #region Local Variables
        protected TextBox textbox;
        GraphicsPath shape;
        GraphicsPath innerRect;
        #endregion
```

```
        #region Constructors
        ///
        public TextBoxRounded()
        {
            InitializingControl();

            textbox = new TextBox();
            textbox.Parent = this;
            Controls.Add(textbox);
            textbox.BorderStyle = BorderStyle.None;
            textbox.Font = Font;
            BackColor = Color.Transparent;
            ForeColor = Color.Black;
            color = Color.White;
            textbox.BackColor = color;
            textbox.TextAlign = textalign;
            Text = null;
            Font = new Font("Century Gothic", 12f);
            Size = new Size(135, 33);
            DoubleBuffered = true;
            textbox.KeyDown += new KeyEventHandler(textbox_KeyDown);
            textbox.TextChanged += new EventHandler(textbox_TextChanged);
            textbox.KeyPress += new KeyPressEventHandler(textbox_KeyPress);
            textbox.MouseDoubleClick +=
                new MouseEventHandler(textbox_MouseDoubleClick);
            Width = 250;
            this.StyleDark.ForeColor = Color.Black;
            this.StyleLight.ForeColor = Color.Black;
        }
        #endregion


        #region Properties
        private HorizontalAlignment textalign;
        ///
        public HorizontalAlignment TextAlign
        {
            get { return textalign; }
            set { textalign = value; textbox.TextAlign = value; }
        }

        private int radius = 15;
        ///
        public int Radius
        {
            get { return radius; }
            set
            {
                if (!autoRadius)
                    radius = value;
                this.Invalidate();
            }
        }

        private bool autoRadius;
        ///
        public bool AutoRaduis
        {
            get { return autoRadius; }
```

```
        set
        {
            autoRadius = value;
            if (value) radius = (int)(0.5 * Height); this.Invalidate();
        }
    }

    private int borderSize = 1;
    ///
    public int BorderSize
    {
        get { return borderSize; }
        set { borderSize = value; this.Invalidate(); }
    }

    ///
    public char PasswordChar
    {
        get { return textbox.PasswordChar; }
        set
        {
            textbox.PasswordChar = value;
            base.Invalidate();
        }
    }

    ///
    public bool UseSystemPasswordChar
    {
        get { return textbox.UseSystemPasswordChar; }
        set
        {
            textbox.UseSystemPasswordChar = value;
            base.Invalidate();
        }
    }

    private Color color;
    ///
    public Color Color
    {
        get { return color; }
        set
        {
            color = value;
            if (color != Color.Transparent)
            {
                textbox.BackColor = color;
            }
            base.Invalidate();
        }
    }

    ///
    public override Color BackColor
    {
        get { return base.BackColor; }
        set { base.BackColor = Color.Transparent; }
    }
```

```
        private Color borderColor = Color.Gray;
        ///
        public Color BorderColor
        {
            get { return borderColor; }
            set { this.borderColor = value; this.Invalidate(); }
        }
        #endregion


        #region Methods
        void InitializingControl()
        {
            base.SetStyle(ControlStyles.SupportsTransparentBackColor, true);
            base.SetStyle(ControlStyles.UserPaint, true);
            base.SetStyle(ControlStyles.ResizeRedraw, true);
        }

        ///
        public void SelectAll()
        {
            textbox.SelectAll();
        }

        public override void ChangeStyle()
        {
            switch (this.Style)
            {
                case Style_e.Dark:
                    this.BackColor = this.StyleDark.BackColor;
                    this.ForeColor = Color.Black;
                    this.Font = this.StyleDark.Font;
                    break;

                case Style_e.Light:
                    this.BackColor = this.StyleLight.BackColor;
                    this.ForeColor = Color.Black;
                    this.Font = this.StyleLight.Font;
                    break;

                default:
                    break;
            }
        }
        #endregion


        #region Events
        private void textbox_KeyPress(object sender, KeyPressEventArgs e)
        {
            base.OnKeyPress(e);
        }

        private void textbox_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
            {
                textbox.SelectAll();
            }
        }
```

```
        private void textbox_TextChanged(object sender, EventArgs e)
        {
            this.Text = textbox.Text;
        }

        private void textbox_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.Control && (e.KeyCode == Keys.A))
            {
                textbox.SelectionStart = 0;
                textbox.SelectionLength = this.Text.Length;
            }
        }

        protected override void OnFontChanged(EventArgs e)
        {
            base.OnFontChanged(e);
            textbox.Font = this.Font;
            base.Invalidate();
        }

        protected override void OnForeColorChanged(EventArgs e)
        {
            base.OnForeColorChanged(e);
            textbox.ForeColor = this.ForeColor;
            base.Invalidate();
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            this.shape = new PaintRectangle(
                (float)base.Width,
                (float)base.Height,
                (float)this.radius,
                0f,
                0f).Path;
            this.innerRect = new PaintRectangle(
                base.Width - 0.5f,
                base.Height - 0.5f,
                (float)this.radius,
                0.5f,
                0.5f).Path;
            if (textbox.Height >= (base.Height - 4))
            {
                base.Height = textbox.Height + 4;
            }
            textbox.Location = new Point(
                this.radius - 5,
                (base.Height / 2) - (textbox.Font.Height / 2));
            textbox.Width = base.Width - ((int)(this.radius * 1.5));
            e.Graphics.SmoothingMode = (SmoothingMode.HighQuality);
            Bitmap bitmap = new Bitmap(base.Width, base.Height);
            Graphics graphics = Graphics.FromImage((Image)bitmap);
            Pen pp = new Pen(borderColor, borderSize);
            e.Graphics.DrawPath(pp, this.shape);

            using (SolidBrush brush = new SolidBrush(color))
            {
                e.Graphics.FillPath((Brush)brush, this.innerRect);
            }
```

```
            Transparency.MakeTransparent(this, e.Graphics);
            base.OnPaint(e);
        }

        protected override void OnTextChanged(EventArgs e)
        {
            base.OnTextChanged(e);
            textbox.Text = this.Text;
        }

        protected override void OnSizeChanged(EventArgs e)
        {
            if (autoRadius)
            {
                radius = (int)(0.5 * Height);
                Invalidate();
            }
        }
        #endregion
    }
}
```

تعتمد الفئة على فئة أخرى موجودة في مجال الأسماء Eng27.CodeBank:



```
using System.Drawing;
using System.Windows.Forms;

namespace Eng27.CodeBank
{
    ///
    internal class Transparency
    {
        ///
        public static void MakeTransparent(Control control, Graphics g)
        {
            Control parent = control.Parent;
            if (parent != null)
            {
                Rectangle rectangle = control.Bounds;
                Control.ControlCollection controls = parent.Controls;
                int index = controls.IndexOf(control);
                Bitmap bitmap = null;
                for (int i = controls.Count - 1; i > index; i--)
                {
                    Control control3 = controls[i];
                    if (control3.Bounds.IntersectsWith(rectangle))
                    {
                        if (bitmap == null)
                        {
                            bitmap = new Bitmap(
                                control.Parent.ClientSize.Width,
                                control.Parent.ClientSize.Height);
                        }
                        control3.DrawToBitmap(bitmap, control3.Bounds);
                    }
                }
```

```
                if (bitmap != null)
                {
                    g.DrawImage(
                        (Image)bitmap,
                        control.ClientRectangle,
                        rectangle,
                        (GraphicsUnit)GraphicsUnit.Pixel);
                    bitmap.Dispose();
                }
            }
        }
    }
}
```

textBoxRounded1

TextBoxRoundedButton

يمكنك – بتطوير الأداة TextBoxRounded – إضافة زر للأداة، كما يمكنك تطوير الأداتين معًا بجعلهما أداة واحدة وتزويد الأداة الجديدة بخاصية تعدادية Enum تعطي المبرمج الخيار لتحديد الأداة التي يرغب بها.



```
using System;
using System.Drawing;
using System.Runtime.InteropServices;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class TextBoxRoundedButton : TextBoxRounded
    {
        #region Local Variables
        private Button button;
        private Panel panel;
        #endregion

        #region DllImports
        [DllImport("user32.dll")]
        private static extern IntPtr SendMessage
            (IntPtr hWnd, int msg, IntPtr wp, IntPtr lp);
        private EventHandler onbuttonClick;
        #endregion
```

```
        #region Constructors
        ///
        public TextBoxRoundedButton() {
            button = new Button();
            panel = new Panel();

            button.Size = new Size(25, textbox.ClientSize.Height);
            button.Dock = DockStyle.Right;
            button.Cursor = Cursors.Default;
            button.Image = Properties.Resources.find;
            button.ImageAlign = ContentAlignment.MiddleCenter;
            button.FlatStyle = FlatStyle.Flat;
            button.ForeColor = Color.White;
            button.BackColor = Color.Transparent;
            button.FlatAppearance.BorderSize = 0;
            button.FlatAppearance.MouseOverBackColor = Color.Transparent;
            button.FlatAppearance.MouseDownBackColor = Color.Transparent;

            panel.Width = linewidth;
            panel.Dock = DockStyle.Right;
            panel.BackColor = this.BorderColor;

            textbox.Controls.Add(panel);
            textbox.Controls.Add(button);

            // طريقة ثانية لربط الأحداث بالأداة
            SendMessage
                (textbox.Handle, 0xd3, (IntPtr)2, (IntPtr)(button.Width << 16));
        }
        #endregion


        #region Properties
        private int linewidth = 2;
        ///
        public int LineWidth
        {
            get { return linewidth; }
            set { linewidth = value; panel.Width = value; }
        }

        ///
        public override RightToLeft RightToLeft
        {
            get { return base.RightToLeft; }
            set
            {
                base.RightToLeft = value;
                if (value == System.Windows.Forms.RightToLeft.Yes)
                {
                    panel.Dock = DockStyle.Left;
                    button.Dock = DockStyle.Left;
                }
                else
                {
                    panel.Dock = DockStyle.Right;
                    button.Dock = DockStyle.Right;
                }
            }
        }
```

```
        ///
        public Image ButtonImage
        {
            get { return button.Image; }
            set
            {
                button.Image = value;
                button.Invalidate();
            }
        }
        #endregion


        #region Events
        private void textbox_KeyPress(object sender, KeyPressEventArgs e)
        {
            base.OnKeyPress(e);
        }
        #endregion


        #region Custom Events
        ///
        public event EventHandler buttonClick
        {
            add
            {
                button.Click += value;
            }
            remove
            {
                button.Click -= value;
            }
        }

        protected virtual void OnbuttonClick(EventArgs e)
        {
            if (onbuttonClick != null)
                onbuttonClick.Invoke(button, e);
        }
        #endregion
    }
}
```

عليك إضافة صورة إلى خصائص المشروع باسم find.

textBoxRoundedButton1

عند النقر على الزر، فإن الحدث buttonClick سيتم تفجيره.

أدوات الرسم Painting

DigitPainter



```
using Eng27.CodeBank.Math;
using Eng27.Components;
using Eng27.Enums;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class DigitPainter : Eng27Control
    {
        #region Constructors
        ///
        public DigitPainter()
        {
            this.Size = new System.Drawing.Size(200, 200);
            this.MouseDown += DigitPainter_MouseDown;
        }
        #endregion


        #region Properties
        private Color gridcolor = Color.Gray;
        ///
        public Color GridColor
        {
            get { return gridcolor; }
            set { gridcolor = value; }
        }

        private Color bitscolor = Color.Red;
        ///
        public Color BitsColor
        {
            get { return bitscolor; }
            set { bitscolor = value; }
        }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public ThemeManager ThemeManager { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Theming_e Theming { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleDark StyleDark { get; set; }
```

```
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleLight StyleLight { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Style_e Style { get; set; }
        #endregion


        #region Events
        void DigitPainter_MouseDown(object sender, MouseEventArgs e)
        {
            Graphics g = this.CreateGraphics();
            g.FillRectangle(
                new SolidBrush(bitscolor),
                Rounding.RoundDown(e.X, 10),
                Rounding.RoundDown(e.Y, 10),
                10,
                10);
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics g = e.Graphics;

            for (int i = 10; i < this.Width; i += 10)
            {
                g.DrawLine(
                    new Pen(gridcolor),
                    new Point(i, 0),
                    new Point(i, this.Height));
            }

            for (int i = 10; i < this.Height; i += 10)
            {
                g.DrawLine(
                    new Pen(gridcolor),
                    new Point(0, i),
                    new Point(this.Width, i));
            }

            base.OnPaint(e);
        }
        #endregion
    }
}
```

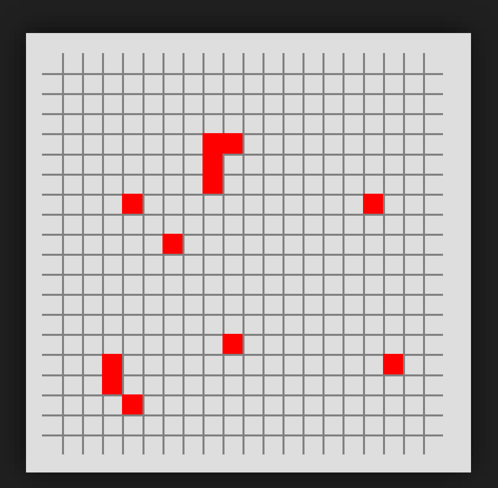
مبدأ هذه الأداة بسيط، شبكة من الخطوط الأفقية والشاقولية، تشكل فيما بينها مربعات، يلوّنها المستخدم بلون معين عندما ينقر على أحد هذه المربعات.

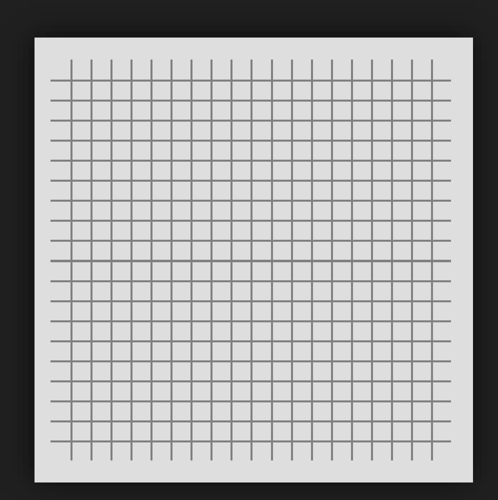
يتم معرفة المربعات بتقريب إحداثيات مؤشر الفأرة لأقرب عشرة، وذلك من خلال الفئة Rounding، التي تقرّب الأعداد لأقرب أكبر أو أصغر عدد صحيح:

```
namespace Eng27.CodeBank.Math
{
    ///
    public static class Rounding
    {
        ///
        public static int RoundUp(int NumberToRound, int BaseNumber)
        {
            if (NumberToRound % BaseNumber == 0) return NumberToRound;
            return (BaseNumber - NumberToRound % BaseNumber) + NumberToRound;
        }

        ///
        public static int RoundDown(int NumberToRound, int BaseNumber)
        {
            return NumberToRound - NumberToRound % BaseNumber;
        }
    }
}
```

يمكنك تطوير هذه الفئة بجعلها إضافة Extension على المتغيرات الصحيحة، وذلك بإضافة الطرق RoundUp وRoundDown للفئة ExtenssionMethods.





كأي كائن رسومي (يعتمد على الكائن Graphics) فإن أي تحديث في منطقة الرسم يؤدي لمسحها؛ وهذا يحدث إذا تغير حجم النافذة أو تم تصغيرها مثلًا. يمكنك تطوير الأداة

بجعلها تحفظ أماكن المربعات الملونة، أو حتى ماهية الألوان فيها، هذا فضلًا عن تخزين إحداثيات وألوان المربعات الملونة في ملفات أو مصدر بيانات وتحميلها منه.

ColorBoard



```
using Eng27.Components;
using Eng27.Enums;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class ColorBoard : Eng27Control
    {
        #region Local Variables
        ButtonFlat[] buttons;
        ButtonFlat selectedbutton;
        int rowsnumber = 2;
        #endregion


        #region Constructors
        ///
        public ColorBoard()
        {
            colors = new Color[15]
            {
                Color.Black,
                Color.White,
                Color.Red,
                Color.Orange,
                Color.Yellow,
                Color.LightGreen,
                Color.Green,
                Color.LightSeaGreen,
                Color.Cyan,
                Color.Azure,
                Color.Blue,
                Color.Violet,
                Color.Magenta,
                Color.MistyRose,
                Color.DarkRed
            };
            this.RecreateButtons();
        }
        #endregion


        #region Properties
        private Color currentcolor = Color.Black;
        ///
        public Color CurrentColor
        {
```

```
            get { return currentcolor; }
            set
            {
                currentcolor = value;
                this.OnCurrentColorChanged(new EventArgs());
            }
        }

        private Color[] colors;
        ///
        public Color[] Colors
        {
            get { return colors; }
            set { colors = value; this.RecreateButtons(); }
        }

        private int colorsize = 25;
        ///
        public int ColorSize
        {
            get { return colorsize; }
            set { colorsize = value; }
        }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public ThemeManager ThemeManager { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Theming_e Theming { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleDark StyleDark { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public StyleLight StyleLight { get; set; }

        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Style_e Style { get; set; }
        #endregion


        #region Methods
        protected override void SetBoundsCore
            (int x, int y, int width, int height, BoundsSpecified specified)
        {
            height = rowsnumber * colorsize;
            width =
                (colors.Length / rowsnumber) *
                colorsize +
                colorsize +
                this.Height +
                10;

            base.SetBoundsCore(x, y, width, height, specified);
        }
```

```
        void RecreateButtons()
        {
            buttons = new ButtonFlat[colors.Length];
            for (int i = 0; i < colors.Length; i++)
            {
                buttons[i] = new ButtonFlat
                {
                    BackColor = colors[i],
                    Size = new Size(colorsize, colorsize),
                    ButtonBorderStyle = ButtonBorderStyle.Solid
                };
            }

            selectedbutton = new ButtonFlat()
            {
                Size = new Size(this.Height, this.Height)
            };

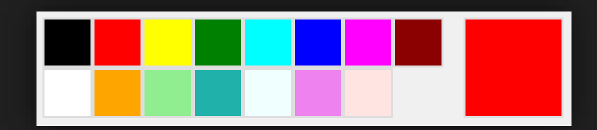
            Controls.Clear();
            int k = 0;
            this.Width =
                (colors.Length / rowsnumber) *
                colorsize +
                colorsize +
                this.Height +
                10;

            for (int i = 0; i < this.Width; i += colorsize)
            {
                for (int j = 0; j < this.Height; j += colorsize)
                {
                    if (k >= colors.Length) break;
                    if (j >= rowsnumber * colorsize) break;
                    Controls.Add(buttons[k]);
                    buttons[k].Left = i; buttons[k].Top = j;
                    buttons[k].Click += ColorBoard_Click;
                    k++;
                }
            }

            Controls.Add(selectedbutton);
            selectedbutton.Left = this.Width - this.Height;
        }
        #endregion


        #region Events
        void ColorBoard_Click(object sender, EventArgs e)
        {
            ButtonFlat button = (ButtonFlat)sender;
            selectedbutton.BackColor = button.BackColor;
            this.CurrentColor = button.BackColor;
        }
        #endregion
```

```
        #region Custom Events
        ///
        public event EventHandler CurrentColorChanged;
        ///
        protected virtual void OnCurrentColorChanged(EventArgs e)
        {
            if (CurrentColorChanged != null)
                CurrentColorChanged.Invoke(this, e);
        }
        #endregion
    }
}
```



جرب أن تستخدم هذه الأداة مع الأداة DigitPainter.

ChartPie

هل تذكر الفصل الخامس؟ الفقرات الأخيرة منه على وجه الخصوص؟ هل تذكر أساليب إنشاء أدوات المستخدم التي ذكرناها في الفصل السادس؟ أسلوب رسم الأداة من الصفر بالذات؟ هذه الأداة ستحوي شيئًا من هذا وذاك، هي ليست بالأسلوب الأخير حرفيًّا، على اعتبار أنها تستند على أداة أخرى ورثت فئتها، إلا أن شيئًا من هذا الأسلوب ستجده هنا.



```
using Eng27.Components;
using Eng27.Enums;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace Eng27.UI
{
    ///
    public class ChartPie : Eng27Control
    {
        #region Local Variables
        Graphics g;
        Brush[] brushes;
        int sum;
        Rectangle rect;
```

```
        GraphicsPath[] paths;
        private bool isPathClicked = false;
        private int PathClickedIndex;
        #endregion


        #region Constructors
        ///
        public ChartPie()
        {
            chartdata = new ChartData[1];
            chartdata[0] = new Eng27.ChartData
            { Name = "New ChartData", Value = 100, BrushColor = Color.Yellow };

            brushes = new SolidBrush[1];
            brushes[0] = new SolidBrush(chartdata[0].BrushColor);
            rect = new Rectangle(10, 10, 150, 150);

            paths = new GraphicsPath[1];

            title = this.Text;
            this.MouseDown += ChartPie_MouseDown;
            this.Size = new Size(320, 250);
        }
        #endregion


        #region Properties
        private int outlinesize = 3;
        ///
        public int OutlineSize
        {
            get { return outlinesize; }
            set { outlinesize = value; }
        }

        private Color outlinecolor = Color.DodgerBlue;
        ///
        public Color OutlineColor
        {
            get { return outlinecolor; }
            set { outlinecolor = value; }
        }

        private string title;
        ///
        public string Title
        {
            get { return title; }
            set
            {
                title = value; OnTitleChanged(new EventArgs());
                this.Invalidate();
            }
        }

        [Browsable(false), EditorBrowsable(EditorBrowsableState.Never)]
        public string Text {
            get; set;
        }
```

```
        private Color titlecolor = Color.Black;
        ///
        public Color TitleColor
        {
            get { return titlecolor; }
            set { titlecolor = value; }
        }

        private float fontsize = 14;
        ///
        public float FontSize
        {
            get { return fontsize; }
            set { fontsize = value; }
        }

        private bool showtitle = true;
        ///
        public bool ShowTitle
        {
            get { return showtitle; }
            set
            {
                showtitle = value;
                this.Invalidate();
            }
        }

        private Color legendColor = Color.Black;
        ///
        public Color LegendColor
        {
            get { return legendColor; }
            set
            {
                legendColor = value;
                this.Invalidate();
            }
        }

        private int legendwitdh = 100;
        ///
        public int LegendWidth
        {
            get { return legendwitdh; }
            set
            {
                legendwitdh = value;
                this.Invalidate();
            }
        }

        private int legendheight = 150;
        ///
        public int LegendHeight {
            get { return legendheight; }
            set { legendheight = value; this.Invalidate(); }
        }
```

```
        private byte legend_selection_color_transparency = 200;
        ///
        public byte LegendSelectionColorTransparency
        {
            get { return legend_selection_color_transparency; }
            set
            {
                legend_selection_color_transparency = value;
                this.Invalidate();
            }
        }

        private bool autolegendsize = true;
        ///
        public bool AutoLegendSize
        {
            get { return autolegendsize; }
            set
            {
                autolegendsize = value;
                this.Invalidate();
            }
        }

        private bool showlegend = true;
        ///
        public bool ShowLegend
        {
            get { return showlegend; }
            set
            {
                showlegend = value;
                this.Invalidate();
            }
        }

        private bool draw_rectangle_onlegend_and_title;
        ///
        public bool DrawRectangleOnLegendAndTitle
        {
            get { return draw_rectangle_onlegend_and_title; }
            set
            {
                draw_rectangle_onlegend_and_title = value;
                this.Invalidate();
            }
        }

        private ChartData[] chartdata;
        ///
        public ChartData[] ChartData
        {
            get { return chartdata; }
            set
            {
                chartdata = value;
                paths = new GraphicsPath[value.Length];
                brushes = new SolidBrush[value.Length];
                for (int i = 0; i < value.Length; i++)
                {
```

```
                brushes[i] = new SolidBrush(value[i].BrushColor);
            }
            this.Invalidate();
        }
    }

    [Browsable(false)]
    [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    public ThemeManager ThemeManager { get; set; }

    [Browsable(false)]
    [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    public Theming_e Theming { get; set; }

    [Browsable(false)]
    [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    public StyleDark StyleDark { get; set; }

    [Browsable(false)]
    [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    public StyleLight StyleLight { get; set; }

    [Browsable(false)]
    [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    public Style_e Style { get; set; }
    #endregion


    #region Methods
    public void Import(string filename) ...
    public void Export(string filename) ...

    int SumOfValues()
    {
        sum = 0;
        foreach (ChartData cd in chartdata)
        {
            sum += cd.Value;
        }
        return sum;
    }
    #endregion


    #region Events
    void ChartPie_MouseDown(object sender, MouseEventArgs e)
    {
        Point mousePt = new Point(e.X, e.Y);

        for (int i = 0; i < paths.Length; i++)
        {
            if (paths[i].IsVisible(mousePt))
            {
                isPathClicked = true;
                PathClickedIndex = i;
                break;
            }
            else { isPathClicked = false; }
        }
        Invalidate(); }
```

```
        protected override void OnPaint(PaintEventArgs e)
        {
            if (chartdata.Length > 0 & SumOfValues() > 0)
            {
                g = e.Graphics;
                g.Clear(this.BackColor);
                g.SmoothingMode = SmoothingMode.AntiAlias;
                int deg1 = 0, deg2 = 0, sum = SumOfValues();
                int stringheight =
                    (int)g.MeasureString(chartdata[0].Name, this.Font).Height;

                if (showtitle)
                {
                    Rectangle textrect = new Rectangle(
                        10,
                        10,
                        this.Width - 25,
                        (int)g.MeasureString
                        (title,
                        new Font(this.Font.FontFamily, fontsize)).Height);

                    StringFormat sf = new StringFormat();
                    sf.Alignment = StringAlignment.Center;
                    if (draw_rectangle_onlegend_and_title)
                    {
                        g.DrawRectangle(Pens.Black, textrect);
                        g.FillRectangle(Brushes.White, textrect);
                    }
                    g.DrawString(title,
                        new Font(
                            this.Font.FontFamily,
                            fontsize),
                            new SolidBrush(titlecolor),
                            textrect, sf);
                    rect.Y = textrect.Height + 50;
                }

                if (showlegend)
                {
                    if (draw_rectangle_onlegend_and_title)
                    {
                        if (autolegendsize)
                            legendheight =
                                ((int)g.MeasureString(
                                chartdata[0].Name,
                                this.Font).Height +
                                5) * chartdata.Length + 10;

                        g.DrawRectangle(
                            Pens.Black,
                            new Rectangle(
                                rect.X + rect.Width + 40,
                                rect.Y,
                                legendwitdh,
                                legendheight));

                        g.FillRectangle(
                            Brushes.White,
                            new Rectangle(
                                rect.X + rect.Width + 40,
```

```
                                rect.Y,
                                legendwitdh,
                                legendheight));
                        }
                }

                int legendy = rect.Y + 10;
                for (int i = 0; i < chartdata.Length; i++)
                {
                    deg2 = chartdata[i].Value * 360 / sum;
                    paths[i] = new GraphicsPath();
                    paths[i].AddPie(rect, deg1, deg2);
                    g.FillPath(brushes[i], paths[i]);
                    g.DrawPath(Pens.Black, paths[i]);
                    deg1 += chartdata[i].Value * 360 / sum;


                    if (showlegend)
                    {
                        g.FillRectangle(
                            brushes[i],
                            new Rectangle(rect.X + rect.Width + 50,
                                legendy,
                                stringheight,
                                stringheight));

                        g.DrawRectangle(
                            Pens.Black,
                            new Rectangle(rect.X + rect.Width + 50,
                                legendy,
                                stringheight,
                                stringheight));

                        g.DrawString(
                            chartdata[i].Name,
                            this.Font,
                            new SolidBrush(legendColor),
                            rect.X + rect.Width + 70,
                            legendy);
                        if (isPathClicked)
                        {
                            if (PathClickedIndex == i)
                                g.FillRectangle(

                                    new SolidBrush(Color.FromArgb(
                                        legend_selection_color_transparency,
                                        chartdata[i].BrushColor)),

                                    rect.X + rect.Width + 70, legendy,

                                    (int)g.MeasureString(
                                    chartdata[i].Name, this.Font
                                    ).Width,

                                    stringheight);
                        }
                        legendy += 5 + stringheight;
                    }
                }
```

```
                if (isPathClicked)
                {
                    Pen outline = new Pen(outlinecolor, outlinesize);
                    g.DrawPath(outline, paths[PathClickedIndex]);
                }
            }
        }
        #endregion


        #region Custom Events
        ///
        public event EventHandler TitleChanged;
        ///
        protected virtual void OnTitleChanged(EventArgs e)
        {
            if (TitleChanged != null)
                TitleChanged.Invoke(this, e);
        }
        #endregion
    }
}
```

تم استنساخ مصفوفة من النوع ChartData واعتبارها أحادية العناصر، مع إسناد قيم أولية لهذا العنصر. وبالمثل لمصفوفة أخرى من النوع SolidBrush. كما تم استنساخ كائنات تمثل منطقة الرسم وربط بعض الأحداث بإجراءاتها.

تملك هذه الفئة طريقة تعطيها مجموع القيم الرقمية لعناصر المصفوفة ChartData، وذلك لاستخدام هذا المجموع لمعرفة النسبة المئوية لكل عنصر من مجموع العناصر.

الفئة ChartData معرفة كما يلي:



```
using System.Drawing;

namespace Eng27
{
    ///
    public class ChartData
    {
        #region Constructors
        ///
        public ChartData() { }
        #endregion

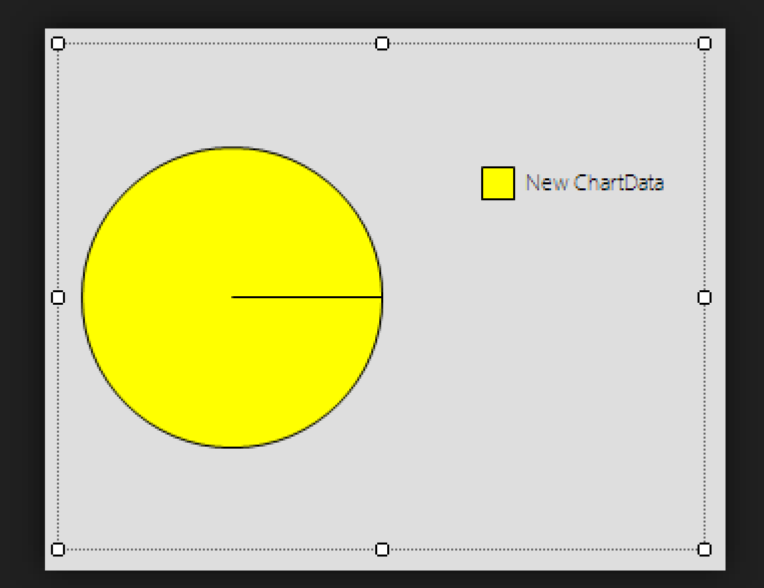

        #region Properties
        private int value;
        ///
        public int Value {
            get { return value; } set { this.value = value; }
        }
```

```
        private string name;
        ///
        public string Name
        {
            get { return name; }
            set { name = value; }
        }

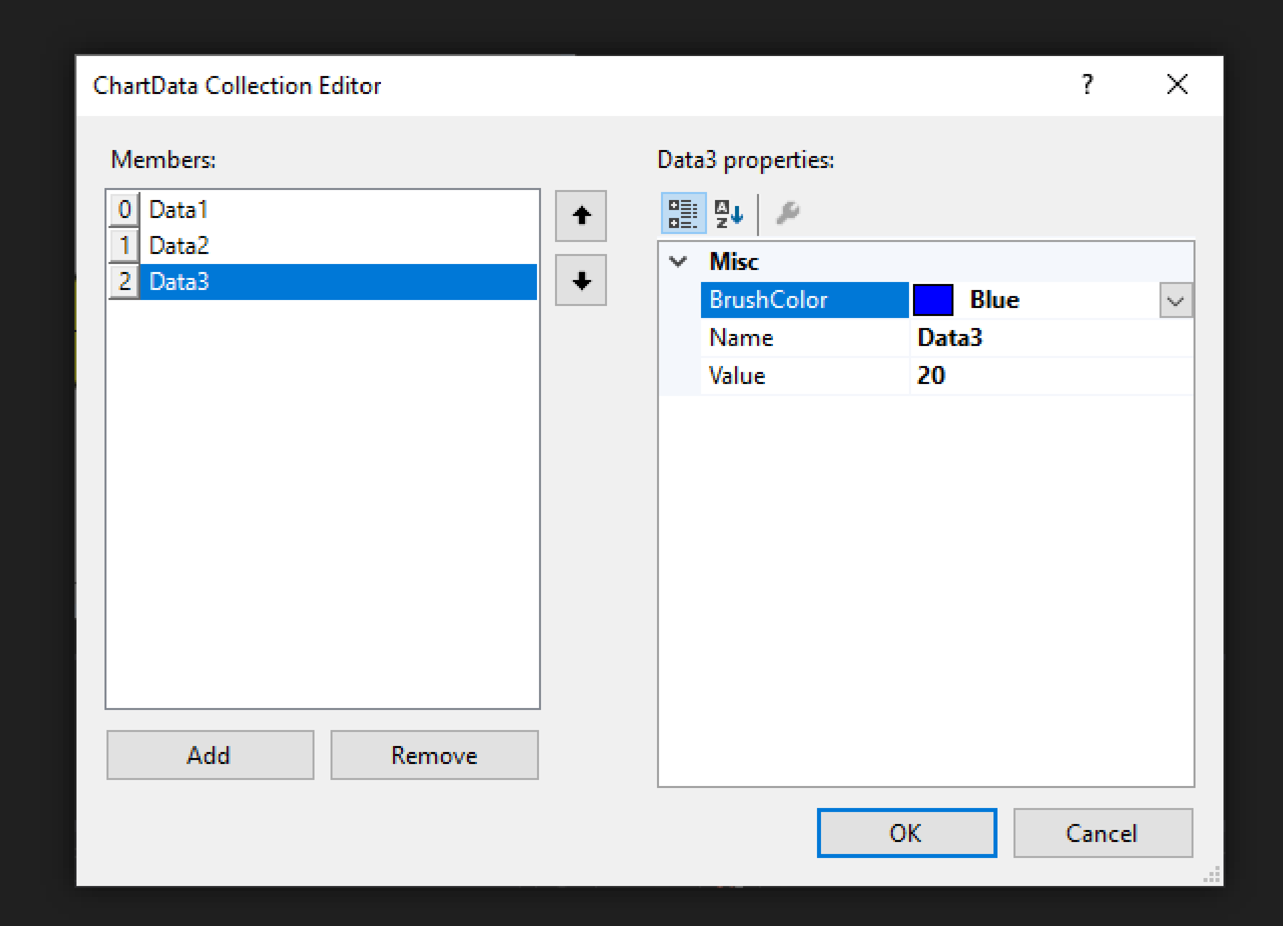
        private Color brushcolor;
        ///
        public Color BrushColor
        {
            get { return brushcolor; }
            set { brushcolor = value; }
        }
        #endregion
    }
}
```

هذه الفئة تمثل بيانات العناصر المكونة للمخطط، يمكنك إنشاء خصائص أخرى لتمثيل بيانات إضافية فيها، حتى لو لم ترغب برسمها أو أنها لا تتعلق مباشرة بالرسم.

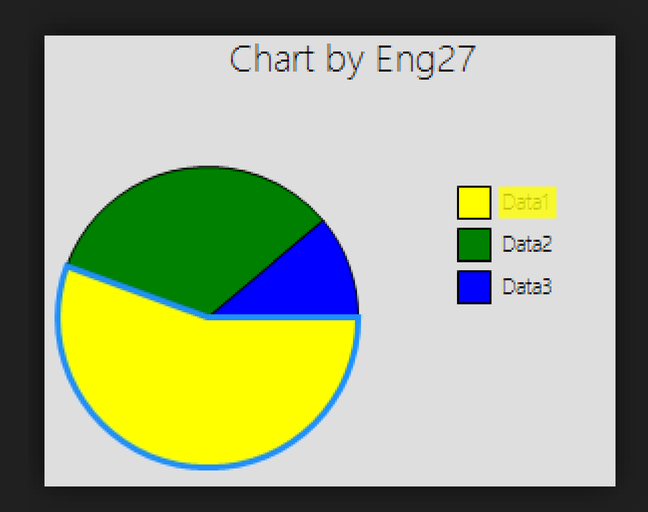
عند إنشاء نسخة من الأداة:

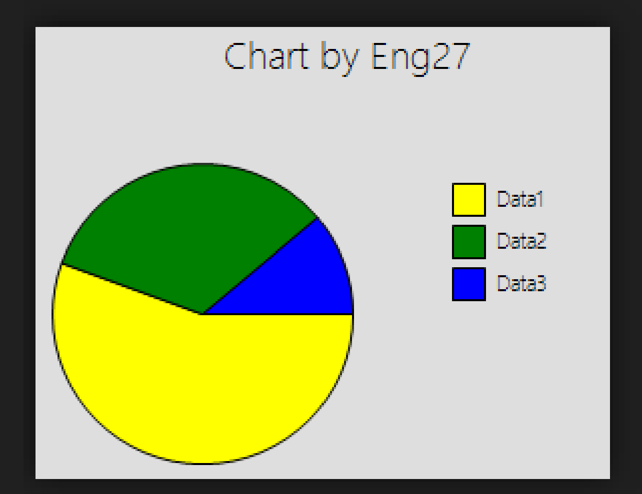


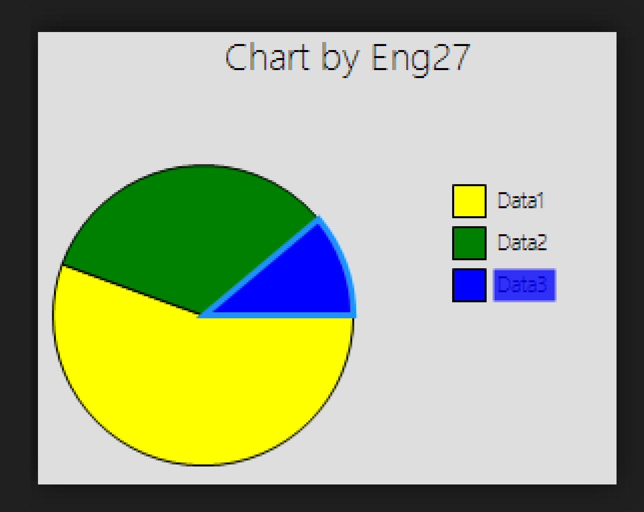
بعد إضافة بعض العناصر:

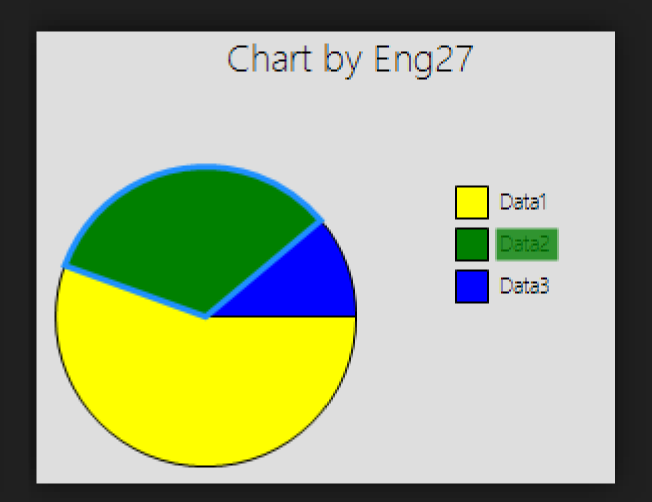


وإسناد قيمة نصية لعنوان المخطط، وتشغيل البرنامج:









المكونات Components

المكونات أشبه بفئات حية، يتعامل معها المبرمج من خلال نافذة الخصائص، أما لو أراد التعامل مع الفئات دون المكونات فلن يحصل – في طور التصميم – على أي حدث أو تنبيه عن اختياراته إذا كانت متناقضة أو تؤدي إلى أخطاء.

يمكننا إطلاق تسمية الأدوات على المكونات، وعندها فإننا نقصد الأدوات غير الرسومية.

ColorTransition

تدمج هذه الأداة لونين معًا، بحيث ترجع لونًا يبدأ من لون أول وينتهي بلون ثانٍ، خلال فترة زمنية معينة. يمكنك جعل الأداة رسومية على مبدأ الأداة ColorBoard، إلا أن كونها غير رسومية أفضل.



```
using Eng27.Interfaces;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace Eng27.Components
{
    ///
    public partial class ColorTransition
        : Component, IEng27Component, IEng27Animation
    {
```

```
#region Local Variables
Timer timer;
int r, b, g;
#endregion


#region Constructors
///
public ColorTransition()
{
    this.InitializeComponent();
    this.InitializeTimer();
}

///
public ColorTransition(IContainer container)
{
    container.Add(this);

    this.InitializeComponent();
    this.InitializeTimer();
}
#endregion


#region Properties
private Color color;
///
[Browsable(false)]
public Color Color
{
    get { return color; }
}

private byte transparency = 150;
///
public byte Transparency
{
    get { return transparency; }
    set { transparency = value; }
}

private Color color1 = Color.Red;
///
public Color Color1
{
    get { return color1; }
    set { color1 = value; r = color1.R; b = color1.B; g = color1.G; }
}

private Color color2 = Color.Yellow;
///
public Color Color2
{
    get { return color2; }
    set { color2 = value; }
}
```

```
        int interval = 20;
        ///
        public int Interval
        {
            get { return interval; }
            set { interval = value; timer.Interval = interval; }
        }

        private bool isanimating = false;
        ///
        [Browsable(false)]
        public bool IsAnimating
        {
            get { return isanimating; }
        }

        private int steps = 5;
        ///
        public int Steps
        {
            get { return steps; }
            set { steps = value; }
        }
        #endregion


        #region Methods
        void InitializeTimer()
        {
            timer = new Timer();

            timer.Interval = interval;
            timer.Tick += timer_Tick;
        }

        ///
        public void Transition()
        {
            r = color1.R; b = color1.B; g = color1.G;
            isanimating = true;
            timer.Enabled = true;
        }

        int RBG(int var, int colorRBG)
        {
            if (var < colorRBG)
                var += steps;
            else if (r > colorRBG)
                var -= steps;
            else
                var = colorRBG;

            if (var < 0) var = 0;
            if (var > 255) var = 255;
            if (Math.Abs(var - colorRBG) < steps) var = colorRBG;
            return var;
        }
        #endregion
```

```
        #region Events
        void timer_Tick(object sender, EventArgs e)
        {
            r = RBG(r, color2.R);
            b = RBG(b, color2.B);
            g = RBG(g, color2.G);

            color = Color.FromArgb(transparency, r, b, g);

            this.OnColorChanged(e);

            if (r == color2.R & b == color2.B & g == color2.G)
            { timer.Enabled = false; isanimating = false; }
        }
        #endregion


        #region Custom Events
        ///
        protected virtual void OnColorChanged(EventArgs e)
        {
            if (ColorChanged != null)
                ColorChanged.Invoke(this, e); // رح يتم تفجير الحدث بالفئة يلي رح تستخدم هالأداة
        }
        #endregion
    }
}
```

تبدأ عملية انتقال اللون عند استدعاء الإجراء Transition، والذي يقوم – أولًا – بضبط قيمة اللون Color بقيمة اللون Color1 وتفعيل المؤقت timer، والذي يقوم باللازم في كل فترة زمنية تحددها الخاصية Interval.

في كل مرة يحدث فيها الحدث timer_Tick يتم تغيير أجزاء اللون (الأحمر والأزرق والأخضر) من خلال التابع RBG، والذي يأخذ وسيطين ويعيد قيمة رقمية، الوسيط الأول هو قيمة جزء اللون الحالي، والوسيط الثاني يمثل قيمة جزء اللون Color2؛ إذ إنه يغير قيمة هذا الجزء وفق الخاصية Step، ويتحقق من كونه ليس سالبا ولا أكبر من 255 (إذ إن أجزاء اللون هي قيمة رقمية من النوع Byte). وبعدها يفجر الحدث OnColorChanged، فإذا كان اللون الحالي Color نفسه اللون المرغوب Color2 يتم إيقاف المؤقت.

أنشئ أداة ما (مثلا ButtonFlat) والمكون ColorTransition، ثم في الحدث ColorChanged أسند قيمة اللون الناتج عن مزج اللونين Color1 وColor2 للون خلفية الأداة التي أنشأتها. وابدأ عملية انتقال اللون بنقرة زر من الأداة:



```
private void colorTransition1_ColorChanged(object sender, EventArgs e)
{
    buttonFlat1.BackColor = colorTransition1.Color;
}

private void buttonFlat1_Click(object sender, EventArgs e)
{
    colorTransition1.Transition();
}
```

ينوب الحدث ColorChanged عن الحدث Tick في المؤقتات، فإنه يحدث في كل مرة يحدث فيها الحدث Tick.

ControlRounding



```
using Eng27.Interfaces;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace Eng27.Components
{
    ///
    public partial class ControlRounding : Component, IEng27Component
    {
        #region Constructors
        ///
        public ControlRounding()
        {
            this.InitializeComponent();
        }

        ///
        public ControlRounding(IContainer container)
        {
            container.Add(this);

            this.InitializeComponent();
        }
        #endregion


        #region Properties
        private int raduis = 10;
        ///
        public int Raduis
        {
            get { return raduis; }
            set { raduis = value; }
        }
```

```
        private Control ctrl;
        ///
        public Control Control
        {
            get { return ctrl; }
            set
            {
                if (!(ctrl is IEng27Control))
                {
                    ctrl = value;
                    RoundControl();
                    ctrl.Paint += ctrl_Paint;
                }
            }
        }
        #endregion


        #region Methods
        private void RoundControl()
        {
            GraphicsPath shape = new GraphicsPath();

            shape = RoundedRect
                (new Rectangle(0, 0, ctrl.Width, ctrl.Height), raduis);

            ctrl.Region = new System.Drawing.Region(shape);

        }

        ///
        public static GraphicsPath RoundedRect(Rectangle bounds, int radius)
        {
            int diameter = radius * 2;
            Size size = new Size(diameter, diameter);
            Rectangle arc = new Rectangle(bounds.Location, size);
            GraphicsPath path = new GraphicsPath();

            if (radius == 0)
            {
                path.AddRectangle(bounds);
                return path;
            }

            // قوس الزاوية العليا اليسارية
            path.AddArc(arc, 180, 90);

            // قوس الزاوية العليا اليمينية
            arc.X = bounds.Right - diameter;
            path.AddArc(arc, 270, 90);

            // قوس الزاوية السفلى اليمينية
            arc.Y = bounds.Bottom - diameter;
            path.AddArc(arc, 0, 90);

            // قوس الزاوية اليسرى السفلية
            arc.X = bounds.Left;
            path.AddArc(arc, 90, 90);
```

```
            path.CloseFigure();
            return path;
        }
        #endregion


        #region Events
        private void ctrl_Paint(object sender, PaintEventArgs e)
        {
            this.RoundControl();
        }
        #endregion
    }
}
```

على سبيل التجربة، أنشئ أداة ما – ولتكن Panel – واجعل حوافها مدورة.

عند تغيير الأداة التي ستدوّر حوافها فإن الأداة القديمة ستحافظ على حوافها المدورة، حتى تقوم بتشغيل المشروع؛ وذلك لأنه لا يوجد ما يجبر الأداة على رسم نفسها إلا تشغيل المشروع. فمن المفيد عندئذ تطوير هذا المكون حتى يعيد الأدوات القديمة لشكلها الأصلي.

ControlShadow



```
using Eng27.Enums;
using Eng27.Interfaces;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace Eng27.Components
{
    ///
    public partial class ControlShadow : Component, IEng27Component
    {
        #region Local Variables
        float trans1, trans2;
        #endregion


        #Constructors#
```

```
#region Properties
private ShadowPosition_e shadowPosition;
///
public ShadowPosition_e ShadowPosition
{
    get { return shadowPosition; }
    set
    {
        shadowPosition = value;

        switch (value)
        {
            case ShadowPosition_e.Top:
                trans1 = 0f; trans2 = -1f; break;
            case ShadowPosition_e.Bottom:
                trans1 = 0f; trans2 = 1f; break;
            case ShadowPosition_e.Left:
                trans1 = -1f; trans2 = 0f; break;
            case ShadowPosition_e.Right:
                trans1 = 1f; trans2 = 0f; break;
        }
    }
}

private int _depth = 15;
///
public int Depth
{
    get { return _depth; }
    set { _depth = value; }
}

private int transparency = 200;
///
public int Tranparency
{
    get { return transparency; }
    set { transparency = value; }
}

private Color _color = Color.Silver;
///
public Color Color
{
    get { return _color; }
    set { _color = value; }
}

private Control parentcontrol;
///
public Control ParentControl
{
    get { return parentcontrol; }
    set
    {
        parentcontrol = value;
        parentcontrol.Resize += parentcontrol_Resize;
        parentcontrol.Paint += parentcontrol_Paint;
    }
}
```

```
        private Control control;
        ///
        public Control Control
        {
            get { return control; }
            set { control = value; parentcontrol = control.Parent; }
        }
        #endregion


        #region Methods
        ///
        public void CreateShadow()
        {
            if (control != null && parentcontrol != null)
            {
                using (Graphics G = parentcontrol.CreateGraphics())
                    drawShadow(
                        G,
                        _color,
                        getRectPath
                        (
                            new Rectangle(control.Left,
                                control.Top,
                                control.Width,
                                control.Height)
                        ),
                        _depth);
            }
        }

        void drawShadow(Graphics G, Color c, GraphicsPath GP, int d)
        {
            Color[] colors =
                getColorVector(c, parentcontrol.BackColor, d).ToArray();
            for (int i = 0; i < d; i++)
            {
                using (Pen pen = new Pen(colors[i], 1f))
                    G.DrawPath(pen, GP);
                G.TranslateTransform(trans1, trans2);
            }
            G.ResetTransform();
        }


        List<Color> getColorVector(Color fc, Color bc, int depth)
        {
            List<Color> cv = new List<Color>();
            float dRed = 1f * (bc.R - fc.R) / depth;
            float dGreen = 1f * (bc.G - fc.G) / depth;
            float dBlue = 1f * (bc.B - fc.B) / depth;
            for (int d = 1; d <= depth; d++)
                cv.Add(Color.FromArgb(transparency, (int)(fc.R + dRed * d),
                  (int)(fc.G + dGreen * d), (int)(fc.B + dBlue * d)));
            return cv;
        }
```

```
        GraphicsPath getRectPath(Rectangle R)
        {
            byte[] fm = new byte[2];
            for (int b = 0; b < 2; b++) fm[b] = 1;
            List<Point> points = new List<Point>();
            switch (shadowPosition)
            {
                case ShadowPosition_e.Top:
                    points.Add(new Point(R.Left, R.Top));
                    points.Add(new Point(R.Right, R.Top));
                    break;
                case ShadowPosition_e.Bottom:
                    points.Add(new Point(R.Left, R.Bottom));
                    points.Add(new Point(R.Right, R.Bottom));
                    break;
                case ShadowPosition_e.Left:
                    points.Add(new Point(R.Left, R.Top));
                    points.Add(new Point(R.Left, R.Bottom));
                    break;
                case ShadowPosition_e.Right:
                    points.Add(new Point(R.Right, R.Top));
                    points.Add(new Point(R.Right, R.Bottom));
                    break;
            }
            return new GraphicsPath(points.ToArray(), fm);
        }
        #endregion


        #region Event
        private void parentcontrol_Paint(object sender, PaintEventArgs e)
        {
            this.CreateShadow();
        }

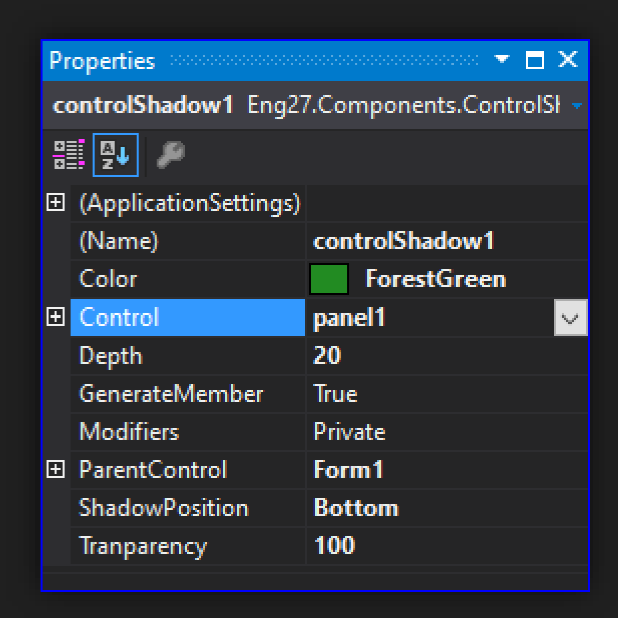
        private void parentcontrol_Resize(object sender, EventArgs e)
        {
            parentcontrol.Invalidate();
        }
        #endregion
    }
}
```

موقع الظل يحدده المعدد ShadowPosition_e:

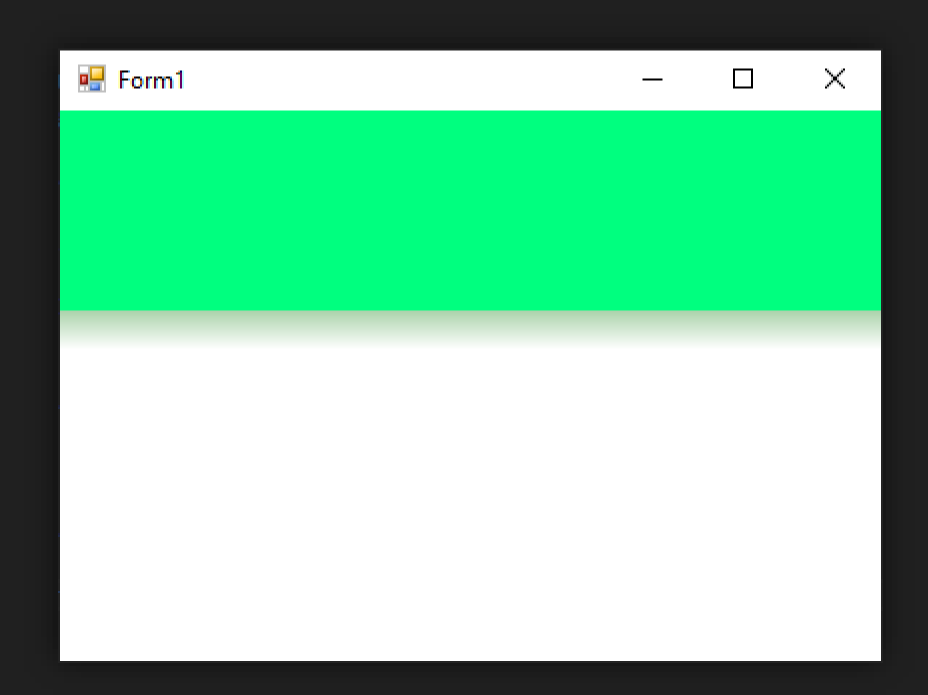


```
namespace Eng27.Enums
{
    ///
    public enum ShadowPosition_e
    {
        Top,
        Bottom,
        Left,
        Right
    }
}
```

اجعل لون خلفية النافذة اللون الأبيض، وأنشئ لائحة Panel واجعل خاصية Dock فيها للأعلى، ولون خلفيتها اللون SpringGreen، وأنشئ المكون ControlShadow واضبط خصائصه كما يلي:



وشغل المشروع:



لاحظ الظل أسفل اللائحة، لا تقلي ما شفته 😒!

ControlScreenshot



```
using Eng27.Exceptions;
using Eng27.Interfaces;
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace Eng27.Components {
    ///
    public partial class ControlScreenshot : Component, IEng27Component
    {
        # Constructors#


        #region Properties
        private string filename;
        /// <summary>
        /// The name of screenshot file, which is BMP file. If you want
        /// file to be saved in the same directory of your application,
        /// just provide the name of BMP file without full path name.
        /// just provide the name of BMP file without full path name.
        /// </summary>
        [Category("Eng27")]
        [Description("The name of screenshot file, which is BMP file. If you want
file to be saved in the same directory of your application, just provide the name
of BMP file without full path name.")]
        public string FileName
        {
            get { return filename; }
            set { filename = value; }
        }
        #endregion


        #region Methods
        /// <summary>
        /// Take screenshot to specific control of the form, including the form
        /// itself.
        /// </summary>
        /// <param name="control">
        /// The control that you want to take screenshot to it.
        /// if Control is the form, just pass "this" keyword.
        /// </param>
        /// <exception cref="Eng27.Exceptions.ScreenshotException"></exception>
        public void TakeScreenshot(Control control)
        {
            try
            {
                using (var bmp = new Bitmap(control.Width, control.Height))
                {
                    Rectangle r =
                        new Rectangle(0, 0, control.Width, control.Height);
                    control.DrawToBitmap(bmp, r);
                    bmp.Save(filename);
                }
            }
```

```
            catch (Exception ex)
            {
                throw new ScreenshotException(ex.Message);
            }
        }
        #endregion
    }
}
```

الاستثناء ScreenshotException ليس إلا الفئة Exception باسم مستعار:



```
using System;

namespace Eng27.Exceptions
{
    class ScreenshotException : Exception
    {
        public ScreenshotException(string msg)
            : base(msg) { }
    }
}
```

أنشئ لائحة Panel، وضع داخلها بعض الأدوات، ثم – عند حدوث حدث ما – اكتب ما يلي:



```
controlScreenshot1.FileName = "screenshot.bmp";
controlScreenshot1.TakeScreenshot(panel1);
```

عند تنفيذ الأكواد السابقة فإنك ستحصل على ملف صورة من النوع bmp بجانب البرنامج، عند فتحه:



يمكنك تطوير المكون بجعله يفتح الصورة بعد إنشاءها أو فتح المجلد الحاوي على الصورة مع الإشارة للصورة، كما يمكنك تزويد هذه الفئة بطرق تغير صيغة الصورة، بالإضافة إلى كتابة صيغة الملف إذا لم يدخلها المبرمج في الخاصية FileName.

وعلى سيرة فتح المجلد الحاوي على الصورة مع الإشارة إليها، فيمكنك ذلك من خلال الطريقة Process.Start، كما يلي:

```
string argument = "/select, \"" + file + "\"";
System.Diagnostics.Process.Start("explorer.exe", argument);
```

ControlExtensionProperty

كيف يعرف الفيجوال ستوديو أن أداة ToolTip تم إضافتها إلى أدوات برنامجك ليضيف خاصية ToolTip On toolTip1 لخصائص بقية الأدوات؟ هل جربت الوصول لهذه الخصائص الإضافية من خلال الأدوات التي تم إضافة الخصائص إليها ولم تحصل على شيء، ثم اكتشفت أنه عليك التعامل مع الأداة المسؤولة عن هذه الخصائص؟

أداتنا هذه ستشرح لك الموضوع.



```
using Eng27.Interfaces;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;

namespace Eng27.Components
{
    ///
    [ProvideProperty("ExtensionProperty", typeof(Control))] //*
    public partial class ControlExtensionProperty
        : Component, IEng27Component, IExtenderProvider
    {
        #region Local Variables
        Dictionary<Control, string> extensions =
                                    new Dictionary<Control, string>();//*
        #endregion


        #Constructors#
```

```
        #region Methods
        ///
        public bool CanExtend(object extendee)//*
        {
            return (extendee is IEng27Control);
        }

        ///
        public void SetExtensionProperty(Control extendee, string value)//*
        {
            if (value.Length == 0)
            {
                extensions.Remove(extendee);
            }
            else
            {
                extensions[extendee] = value;
            }
        }

        ///
        [DisplayName("ExtensionProperty")]//*
        [ExtenderProvidedProperty()]//*
        [DefaultValue("Default Value")]//*
        public string GetExtensionProperty(Control extendee)//*
        {
            if (extensions.ContainsKey(extendee))
            {
                return extensions[extendee];
            }
            else
            {
                return string.Empty;
            }
        }
        #endregion
    }
}
```

الأسطر التي تنتهي بـ * ضرورية، هي مسؤولة عن إضافة الخاصية إلى الأدوات.

ضع في ذهنك أن الخصائص التي يتم إضافتها إلى الأدوات هي خصائص وهمية لا وجود لها في الأدوات التي تمت إضافتها إليها؛ لذلك لا يمكنك الوصول إليها – أعني الخصائص – من خلال الأدوات المستهدَفة. كما أنها خصائص تتعلق بوقت التصميم Design-Time، لن تجدها وقت التنفيذ Run-Time. وهذا كله لأنها ليست إلا عناصر مجموعة من البيانات (قاموس Dictionary) تحوي الأداة كفهرس، والقيمة المرتبطة بهذا الفهرس.

هذا المثال ليس تطبيقًا واقعيًا، ولا يمثل شيئا ملموسًا، فهو ليس إلا شرحًا أو إيجازًا لنوع الأدوات الذي نتناوله في هذه الفقرة. وأتوقع أنك أحسست بشيء من هذا من اسم الأداة ControlExtensionProperty واسم الخاصية ExtensionProperty.

يمكن التعامل مع جميع ما تملكه الأداة من سجلات بالشكل التالي على سبيل المثال (هناك أساليب أخرى يمكن التعامل بها، تدور الفكرة حول القواميس Dictionaries):



```
foreach (Control c in extensions.Keys)
{
    ((IEng27Control)c).Style = style;
    ((IEng27Control)c).StyleDark = styledark;
    ((IEng27Control)c).StyleLight = stylelight;
}
```

لو أردت الوصول مباشرة للقيم التي يحويها القاموس، الخاصية Values هي غريمك.

ControlDrag

يمكنك تغيير موضع الأدوات من خلال التحكم بموقعها Location، وبشكل مشابه فإن التحكم بحجمها يكون عن طريق الخاصية Size. ولجعل العملية أكثر واقعية فإن أحداث الفأرة ستساعدك في التحكم بمواقع الأدوات بشكل آني.



```
using Eng27.Interfaces;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace Eng27.Components
{
    ///
    public partial class ControlDrag : Component, IEng27Component
    {
        #region Local Variables
        Point previousLocation;
        bool dragging = false;
        #endregion


        #Constructors#
```

```
        #region Properties
        private bool allowdragging = true;
        ///
        public bool AllowDragging
        {
            get { return allowdragging; }
            set { allowdragging = value; }
        }

        private Control control;
        ///
        public Control Control
        {
            get { return control; }
            set
            {
                if (value != null)
                {
                    control = value;
                    control.MouseDown += activeControl_MouseDown;
                    control.MouseMove += activeControl_MouseMove;
                    control.MouseUp += activeControl_MouseUp;
                }
            }
        }
        #endregion


        #region Events
        void activeControl_MouseUp(object sender, MouseEventArgs e)
        {
            if (allowdragging)
            {
                control.Cursor = Cursors.Default;
                control = null; dragging = false;
            }
        }

        void activeControl_MouseMove(object sender, MouseEventArgs e)
        {
            if (allowdragging)
            {
                if (dragging)
                {
                    if (control == null || control != sender)
                        return;

                    var location = control.Location;
                    location.Offset(e.Location.X - previousLocation.X,
                        e.Location.Y - previousLocation.Y);
                    control.Location = location;
                }
            }
        }

        void activeControl_MouseDown(object sender, MouseEventArgs e)
        {
            if (allowdragging)
            {
                control = sender as Control;
```

```
                previousLocation = e.Location;
                control.Cursor = Cursors.Hand;
                dragging = true;
            }
        }
        #endregion
    }
}
```

المميز بهذه الأداة أنها تستطيع التحكم بالنوافذ حتى، فحتى النوافذ عديمة الحواف – والتي لا يمكنك نقلها بشكل مباشر ضمن الشاشة – يمكن نقلها من خلال هذه الأداة.

FileModel

تساعدك هذه الأداة بإنشاء أنظمة ملفات لا يمكن تصفحها مباشرة من خلال محرر النصوص، وإنما تحتاج محرّرًا خاصًّا لها تبرمجه أنت. الموضوع ليس ببساطة تغيير لواحق الملفات مع الحفاظ على المحتوى النصي للملف ذاته، الأمر أعقد من ذلك.

لكن بالمقابل، فإن مبدأ العملية ليس معقدًّا لدرجة اختراع خوارزمية ملفات جديدة، كل ما في الموضوع أن هذه الفئة تخزن الملفات مع بعضها على شكل مصفوفة من الملفات، ضمن ملف واحد. إذ تعتبر أن كل ملف هو مصفوفة من البايتات [] byte، فإذا أنشأنا مصفوفة ثنائية [][] byte [1] فإننا نشنئ مصفوفة من الملفات (وكأننا نقول [] File).

تضم هذه الفئة بين أسطرها طرقًا مختلفة لإنشاء الملفات (تجميعها ضمن ملف واحد وفكها (تحويل الملف الواحد إلى مصفوفة ثنائية من البايتات)، وهي تدعم تحويل الملفات أو العبارات النصية إلى ملفات. فمهما كان نوع الملفات الذي ترغب بتضمينه ضمن نظام ملفاتك، حوله لمصفوفة من البايتات، وهذا ما تقوم به هذه الأداة وتختصر عليك المشوار.

```
using Eng27.Enums;
using Eng27.Exceptions;
using Eng27.Interfaces;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
```

[1] في الواقع فإن المصفوفات الثنائية في البرمجة 2D Arrays ليست إلا شعاعًا من الأشعة، أي أن الشعاع نفسه (المصفوفة أحادية الأبعاد) يتم وضعه ضمن شعاع من نفس النوع. وبالمثل المصفوفات متعددة الأبعاد. أما المصفوفات (أو المتاريس) الرياضية Matrices، فهي بعيدة جدًا عن المصفوفات البرمجية.

```
using System.Runtime.Serialization.Formatters.Binary;

namespace Eng27.Components
{
    ///
    public partial class FileModel : Component, IEng27Component
    {
        #region Local Variables
        byte[][] filesasbytes;
        #endregion


        #Constructors#


        #region Properties
        private List<string> files = new List<string>();
        ///
        [Browsable(false)]
        public List<string> Files
        {
            get { return files; }
            set { files = value; }
        }

        Dictionary<string, string> strings = new Dictionary<string, string>();
        ///
        [Browsable(false)]
        public Dictionary<string, string> Strings
        {
            get { return strings; }
            set { strings = value; }
        }

        private string outputDiroctory;
        ///
        public string OutputDircetory
        {
            get { return outputDiroctory; }
            set { outputDiroctory = value; }
        }

        private string filename;
        ///
        public string FileName
        {
            get { return filename; }
            set { filename = value; }
        }
        #endregion


        #region Methods
        byte[][] FilesToArrays()
        {
            List<byte[]> fileasbites;
            fileasbites = new List<byte[]>();

            string filenames = "";
            foreach (string s in files)
```

```
        {
            filenames += s + Environment.NewLine;
        }

        byte[] filenamesbytes = null;
        using (var ms = new MemoryStream())
        {
            TextWriter tw = new StreamWriter(ms);
            tw.Write(filenames);
            tw.Flush();
            ms.Position = 0;
            filenamesbytes = ms.ToArray();
        }

        fileasbites.Add(filenamesbytes);

        for (int i = 0; i < files.Count; i++)
        {
            fileasbites.Add(File.ReadAllBytes(files[i]));
        }

        return fileasbites.ToArray();
    }

    byte[][] StringsToArrays()
    {
        List<byte[]> stringasbites;
        stringasbites = new List<byte[]>();

        string stringnames = "";
        foreach (string s in strings.Keys)
        {
            stringnames += s + Environment.NewLine;
        }

        byte[] stringnamesasbites = null;
        using (var ms = new MemoryStream())
        {
            TextWriter tw = new StreamWriter(ms);
            tw.Write(stringnames);
            tw.Flush();
            ms.Position = 0;
            stringnamesasbites = ms.ToArray();
        }

        stringasbites.Add(stringnamesasbites);

        for (int i = 0; i < strings.Count; i++)
        {
            using (var ms = new MemoryStream())
            {
                TextWriter tw = new StreamWriter(ms);
                tw.Write(strings.ElementAt(i).Value);
                tw.Flush();
                ms.Position = 0;
                stringasbites.Add(ms.ToArray());
            }
        }
        return stringasbites.ToArray();
    }
```

```
        ///
        public Stream SerializeStrings(bool CreateFile)
        {
            try
            {
                filesasbytes = StringsToArrays();
                var model = new DataModel
                {
                    Files = filesasbytes,
                };
                var formatter = new BinaryFormatter();
                Stream filemodel = new MemoryStream();
                formatter.Serialize(filemodel, model);
                if (CreateFile)
                {
                    if (filename == "")
                        throw new FileModelException
                            (FileModelExceptionType_e.FileNameNotSpecified);
                    if (!File.Exists(filename))
                        throw new FileModelException
                            (FileModelExceptionType_e.FileNotFound);

                    using (var output = File.Create(filename))
                    {
                        formatter.Serialize(output, model);
                    }
                }
                return filemodel;
            }
            catch (Exception ex) { throw new Exception(ex.Message, ex); }
        }

        ///
        public Dictionary<string, string> DeserializeStrings()
        {
            try
            {
                var formatter = new BinaryFormatter();
                List<string> lines = new List<string>();
                Dictionary<string, string> res = new Dictionary<string, string>();

                if (filename == "")
                    throw new FileModelException
                        (FileModelExceptionType_e.FileNameNotSpecified);
                if (!File.Exists(filename))
                    throw new FileModelException
                        (FileModelExceptionType_e.FileNotFound);

                using (var input = File.OpenRead(filename))
                {
                    var model = (DataModel)formatter.Deserialize(input);

                    byte[] stringnames = model.Files[0];

                    using (var ms = new MemoryStream(stringnames))
                    {
                        TextReader tr = new StreamReader(ms);
                        string s = tr.ReadToEnd();
                        foreach (string ss in s.Split
                            (new string[] { "\r\n" }, StringSplitOptions.None))
```

```
                    {
                        lines.Add(ss);
                    }
                }

                for (int i = 1; i < model.Files.Length; i++)
                {
                    using (var ms = new MemoryStream(model.Files[i]))
                    {
                        TextReader tr = new StreamReader(ms);
                        res.Add(lines[i - 1], tr.ReadToEnd());
                    }
                }
                return res;
            }
        }
        catch (Exception ex) { throw new Exception(ex.Message, ex); }
    }

    Stream serializefiles(bool CreateFile)
    {
        try
        {
            filesasbytes = FilesToArrays();
            var model = new DataModel
            {
                Files = filesasbytes,
            };
            Stream stream = new MemoryStream();
            if (CreateFile)
            {
                var formatter = new BinaryFormatter();
                formatter.Serialize(stream, model);
                using (var output = File.Create(filename))
                {
                    formatter.Serialize(output, model);
                }
            }

            return stream;
        }
        catch (Exception ex) { throw new Exception(ex.Message, ex); }
    }

    ///
    public Stream SerializeFiles(bool CreateFile, List<string> FilePaths)
    {
        files = FilePaths;
        return serializefiles(CreateFile);
    }

    ///
    public Stream SerializeFiles(bool CreateFile)
    {
        return serializefiles(CreateFile);
    }
```

```
        ///
        public DataModel DeserializeFiles(bool CreateFiles)
        {
            try
            {
                var formatter = new BinaryFormatter();
                using (var input = File.OpenRead(filename))
                {
                    var model = (DataModel)formatter.Deserialize(input);
                    if (CreateFiles)
                        DeserializeFilesInDir(model);
                    return model;
                }
            }
            catch (Exception ex) { throw new Exception(ex.Message, ex); }
        }

        void DeserializeFilesInDir(DataModel m)
        {
            if (Directory.Exists(outputDirectory))
            {
                byte[] filenames = m.Files[0];

                using (var ms = new MemoryStream(filenames))
                {
                    TextReader tr = new StreamReader(ms);
                    string s = tr.ReadToEnd();
                    List<string> lines = new List<string>();
                    foreach (string ss in s.Split
                        (new string[] { "\r\n" }, StringSplitOptions.None))
                    {
                        lines.Add(ss);
                    }

                    for (int i = 1; i < m.Files.Length; i++)
                    {
                        File.WriteAllBytes
                            (outputDirectory + "\\" + lines[i - 1], m.Files[i]);
                    }
                }
            }
            else
                DeserializeFilesHere(m);
        }

        void DeserializeFilesHere(DataModel m)
        {
            byte[] filenames = m.Files[0];

            using (var ms = new MemoryStream(filenames))
            {
                TextReader tr = new StreamReader(ms);
                string s = tr.ReadToEnd();
                List<string> lines = new List<string>();
                foreach (string ss in s.Split
                    (new string[] { "\r\n" }, StringSplitOptions.None))
                {
                    lines.Add(ss);
                }
```

```
                for (int i = 1; i < m.Files.Length; i++)
                {
                    File.WriteAllBytes(lines[i - 1], m.Files[i]);
                }
            }
        }
        #endregion
    }
}
```

إذا كان ما ترغب بتضمينه ضمن نظام الملفات هو مجموعة من الملفات، فالطريقة FilesToArrays يتم استدعاءها، لتحول الملفات – بعد قراءتها إلى مصفوفة من البايتات. أما إذا أردت تحويل النصوص فإن الطريقة StringsToArrays تقوم بذلك (فعليًّا هي مثل الطريقة الأولى إذا جعلتها تحول الملفات النصية إلى مصفوفات). كلا الطريقتين تنشئان في البداية مصفوفة من البايتات تحوي أسماء الملفات ضمن الملف الناتج، أي أن الملف الذي سننشئه وعلى اعتباره يحوي عدة ملفات سيحوي في البداية على ملف (مصفوفة من البايتات) فيه أسماء الملفات ضمن هذا الملف، فالعنصر الأول من مصفوفة الملفات دائما – في ملفات هذه الأداة – هو فهرس لمحتويات الملف. ولولا وجود هذا العنصر الأول لما تمكنا من معرفة أسماء الملفات وأنواعها عند فكها. فإذا قمت بتضمين صورتين من نوع jpg وملف صوت من النوع mp3 ضمن ملف ما فإن محتوياته ستكون أربعة ملفات (مصفوفة ثنائية، شعاع من الأشعة، وعددها في هذه الحالة أربعة)، الأول يمكن استعراضه نصيًّا وفيه ما يشبه هذا:

File1.jpg

File2.jpg

File3.mp3

أما بقية العناصر فمحتوياتها تتبع لنوع الملفات الذي تعود إليها.

الفئة DataModel هي فعليًّا جوهر الأداة، فهي عبارة عن مصفوفة الملفات:



```
using System;

namespace Eng27
{
    ///
    [Serializable]
    public class DataModel
    {
        public byte[][] Files { get; set; }
    }
}
```

أما استثناء فئة هذه الأداة فهو معرف بالفئة FileModelException:



```
using Eng27.Enums;
using System;

namespace Eng27.Exceptions
{
    ///
    public class FileModelException : Exception
    {
        string msg = "";
        ///
        public FileModelException(FileModelExceptionType_e exception_type)
            : base("File not found, or Filename not specified")
        {
            this.ExceptionType = exception_type;
            if (exception_type == FileModelExceptionType_e.FileNameNotSpecified)
                msg = "FileName not specified.";
            else if (exception_type == FileModelExceptionType_e.FileNotFound)
                msg = "File not found.";
        }

        ///
        public FileModelExceptionType_e ExceptionType { get; private set; }

        public override string Message
        {
            get
            {
                return this.msg;
            }
        }
    }
}
```

والذي يتم تحديد نوعه من خلال المعدد FileModelExceptionType_e:

```
namespace Eng27.Enums
{
    ///
    public enum FileModelExceptionType_e
    {
        FileNameNotSpecified,
        FileNotFound
    }
}
```

وكمثال، أنشئ نسخة من الأداة FileModel واستخدم الكود التالي لإنشاء ملف وفق نظام الملفات التي تعطيك إياه الأداة:



```
// الملفات موجودة في نفس مسار المشروع
// وإلا، عليك التصريح عن المسار كاملاً
fileModel1.Files.Add("Audio file.mp3");
fileModel1.Files.Add("Video file.mp4");
fileModel1.Files.Add("Image file.bmp");

fileModel1.FileName = "My file.mf";

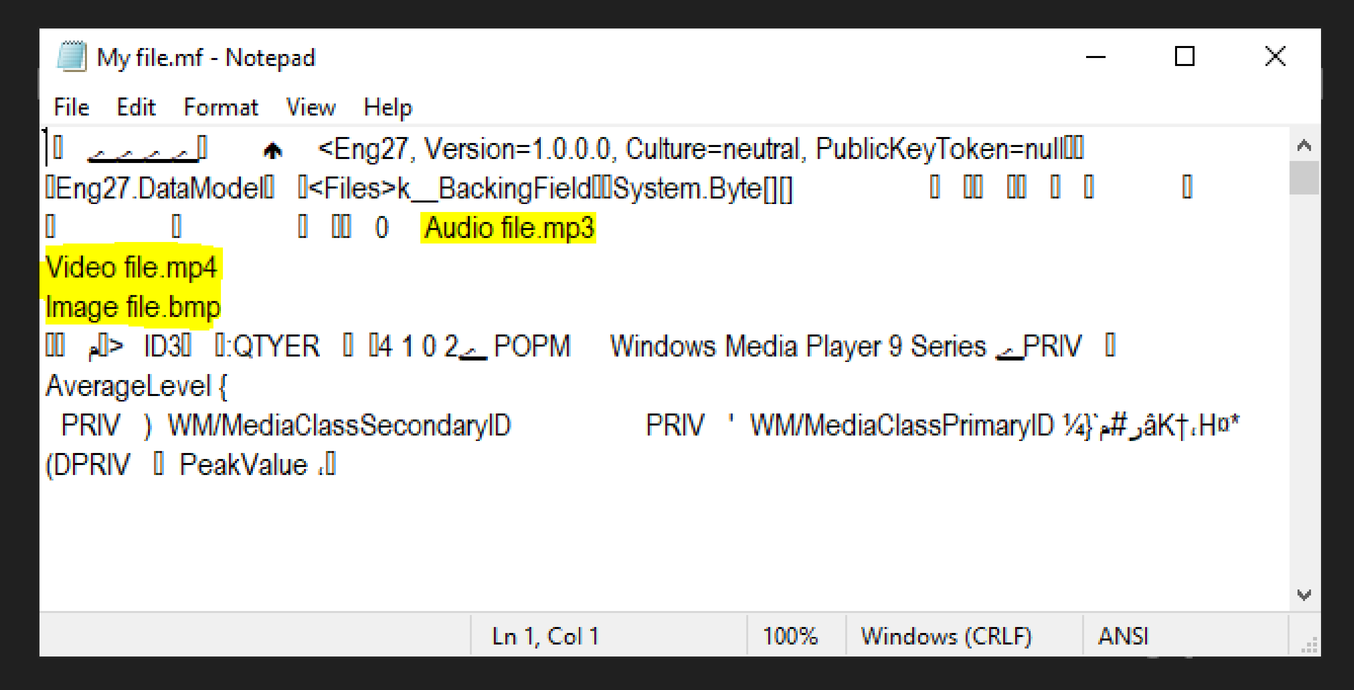
fileModel1.SerializeFiles(true);
```

الملف الناتج سيكون بحجم مجموع الملفات التي يحويها، بالإضافة لحجم العنصر الأول الذي يحوي فهرسًا بالملفات التي يحويها:

My file.mf	12/1/2020 08:40	MF File	10,272 KB

يمكنك أيضًا إنشاء كائن من النوع Stream وإسناد القيمة المعادة من الطريقة SerializeFiles إليه، للتحكم بالملفات مباشرة من خلال الكود عوضًا إنشاءها ضمن القرص الصلب. ومع ذلك يمكنك بالحالتين – إذا أردت التعامل مع كائن Stream أو لا – إنشاء الملفات أو عدم إنشاءها.

إذا حاولت فتح الملف بغير البرنامج الذي يستطيع قراءته – والذي يفترض أن يكون من برمجتك، على اعتبارك أنت صاحب نظام الملفات – كقيمة نصية فإنك ستحصل على مجموعة كبيرة من الطلاسم، وفي بدايتها كلام كهذا:



لاحظ أن بداية الملف يمكن فهمه بطريقة أو بأخرى، وهو يمثل أول عنصر من عناصر المصفوفة التي تمثل الملفات المحتواة ضمن الملف، وهذا العنصر ما هو إلا مصفوفة من العناصر النصية، من خلاله يمكنك معرفة عدد الملفات وأسماءها ونوعها، لتستخدم هذه المعلومات عند فك الملف إلى مكوناته الأساسية، لاحظ:

```
fileModel1.FileName = "My file.mf";
fileModel1.OutputDircetory = "d:\Output dir";

fileModel1.DeserializeFiles(true);
```

يقوم التابع DeserializeFiles بفك الملف وتحويله إلى مصفوفة من الملفات، وإنشاء هذه الملفات في المسار المعرف من خلال الخاصية OutputDirectory، على اعتبار تم تمرير القيمة true إلى التابع.

إذا أربكتك أسطر الفئة FileModel فإني أحيلك إلى الكود الأصلي [1] الذي اعتمدت عليه لتطوير هذه الأكواد، والذي كان محتواه:



```
using System;
using System.Windows.Forms;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

namespace SerializingTest
{
    public partial class Form1 : Form
    {
        public Form1() ...

        void Serialize()
        {
            var model = new MyModel
            {
                Images = new[]
                {
                    File.ReadAllBytes(@"d:\work\image1.png"),
                    File.ReadAllBytes(@"d:\work\image2.png"),
                    File.ReadAllBytes(@"d:\work\image3.png"),
                }
            };

            var formatter = new BinaryFormatter();
            using (var output = File.Create("images.dat"))
            {
                formatter.Serialize(output, model);
            }
        }

        void Deserialize()
        {
            var formatter = new BinaryFormatter();
            using (var input = File.OpenRead("images.dat"))
            {
                var model = (MyModel)formatter.Deserialize(input);
            }
        }
    }

    [Serializable]
    public class MyModel
    {
        public byte[][] Images { get; set; }
    }
}
```

[1] راجع السؤال من موقع StockoverFlow: تخزين صور عديدة في ملف واحد
https://stackoverflow.com/questions/12387978/c-sharp-storing-multiple-images-in-a-file

وهنا نقطة مهمة، قد تستطيع اختصار فكرة ما ببضعة أسطر أو عشرات قليلة منها، إلا أن تنفيذها في الواقع وإنشاء التفاصيل الثانوية فيها قد يجعل عدد أسطرها يصل للمئات أو الآلاف. الفرق بين هذه الأسطر القليلة وأسطر فئتنا، أن فئتنا فيها أتمتة لعمل هذه الأسطر.

أدوات مقترحة

كثيرة هي الأدوات التي كنت أنوي تصميمها وتضمينها ضمن المكتبة، إلا أني لم أصل لمبتغاي؛ لضيق الوقت وعدم اهتدائي لتصاميم مقبولة لبعض الأدوات.

كما أنك ستصل لأفكار كثيرة لتصاميم تنفعك في برامجك بتطوير الأفكار الموجودة في هذا الباب – وهذا الفصل خصوصًا – أو بالتفكير في الأدوات الموجودة هنا وهناك، أو قد تجد أفكارًا لأدوات لم يسبقك أحد عليها، خصوصًا إذا كان اختصاصك لا علاقة له بالمجال المعلوماتي والحوسبي، وحاولت برمجة وأتمتة بعض علوم اختصاصك.

في هذه الفقرة سأضع بعض الأفكار التي يمكنك تطويرها والاستفادة منها، وعدم تصميمي إياها لا يعني صعوبتها، فالوقت أضيق من أن تجمع كل شيء في كتاب – أو فصل – واحد!

عنوان قابل لتغيير قيمته النصية:

يمكنك دمج الأداة TextBox بالأداة Label، بإخفاء صندوق النص وإظهار العنوان، فإذا نقر المستخدم مرتين على العنوان – أو قام بشيء ما، الضغط على مفتاح مثلًا – ظهر صندوق النص مكانه، وقيمته النصية مساوية للقيمة النصية للعنوان، فإذا غيرها المستخدم ونقر على مفتاح ما (مفتاح Enter مثلا) اختفى صندوق النص وظهر العنوان مكانه وقد أخذ قيمة نصية جديدة.

صندوق نص مخصص لكلمات السر:

يمكنك إنشاء أداة مستخدم User Control فيها زر وصندوق نص، فإذا ضغط المستخدم واستمر بالضغط على الزر ظهرت كلمة السر، وعند رفع الضغط تعود لوضعها الأصلي.

صندوق نص فيه عنوان متحرك:

البريد الإلكتروني أو الهاتف

البريد الإلكتروني أو الهاتف

مثال آخر:

Your passcode

Your passcode

صندوق نص رقمي:

على مبدأ الأداة UpDownNumeric يمكنك إنشاء صندوق نص يعيد قيمة رقمية بدل قيمة نصية. والأكثر من ذلك، يمكنك إنشاء خاصية لتحديد نوع البيانات المراد إرجاعها، أو طريقة لذلك.

صندوق نص يدعم الواحدات الفيزيائية:

بتطوير أداة صندوق النص الرقمي يمكنك تضمين قائمة منسدلة بالواحدات الفيزيائية التي تمثلها الأداة، والتي يتم إدخالها عن طريق مصفوفة نصية بالواحدات الفيزيائية، تمامًا كإسناد مصدر للإكمال التلقائي.

كما يمكنك تطوير الأداة أكثر من ذلك دون الاعتماد على القوائم المنسدلة، وذلك بجعل صندوق النص نفسه يقبل الواحدة الفيزيائية ويفهمها، بهذا الشكل "5 m/s"، والتي تفهم منها الأداة أن القيمة هي 5 والواحدة هي "m/s"، بغض النظر عن ماهية هذه القيمة (مسافة، سرعة، تسارع، عزم، استطاعة، ...). لكنك تحتاج لهذا لمعالجة النصوص.

مصحح لغوي SpellChecker:

مكون Component يأخذ مصفوفة نصية فيها الكلمات الصحيحة (على شكل قاموس من الكلمات)، وفيه طريقة Method تأخذ قيمة نصية كوسيط (أو أداة ما) وتبحث عن الكلمات فيها. يوجد أيضًا لائحة List بالكلمات التي تخالف مصفوفة الكلمات الصحيحة ورقم سطرها.

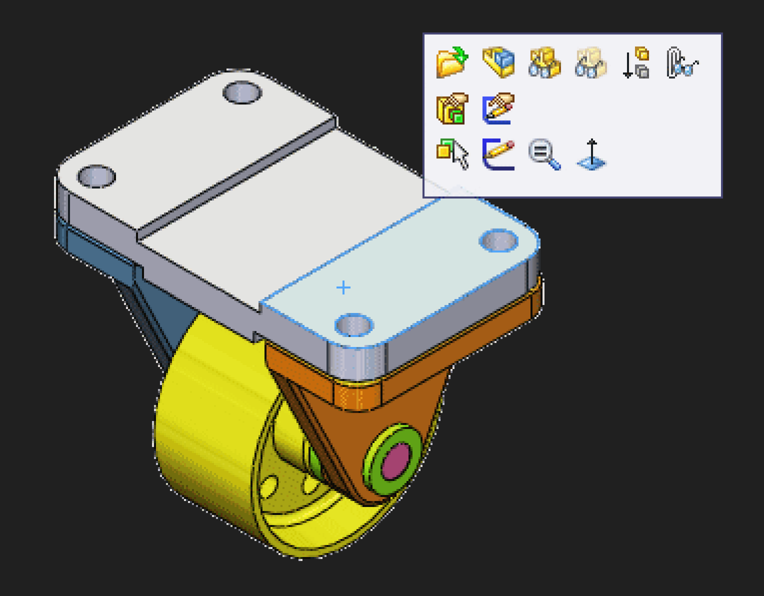
صندوق إدخال InputBox:

على غرار صندوق الرسالة MessageBox يمكنك إنشاء صندوق إدخال، عبارة عن نافذة صغيرة بزر إغلاق فقط، لها عنوان ووصف، وصندوق نص للإدخال، وزر لقبول القيمة، وأزرار أخرى وفق الحاجة (يختار المبرمج ما يود أن يظهره منها). كما أنك بحاجة لإنشاء خاصية ما لاحتواء القيمة المدخلة، مثلًا Value.

عند استدعاء الطريقة InputBox.Show، وإذا كانت قيمة صندوق الحوار OK، فإنه يمكن الحصول على قيمة صندوق الإدخال هذا بالوصول للخاصية Value (والتي ستكون لا شيء null إذا لم تكن قيمة صندوق الحوار OK).

قائمة منبثقة بأدوات متعلقة بالسياق ContextualMenu:

عند النقر على مكان ما ضمن التطبيق، تظهر قائمة على شكل لائحة فيها أيقونات، تتعلق هذه القائمة بالسياق الحالي. فمثلا: في برنامج SolidWorks (برنامج تصميم ميكانيكي) عند النقر على جزء من التصميم تظهر قائمة بالأوامر المتعلقة بالسياق المنقور عليه، وعند النقر على غيره تظهر بأوامر أخرى.

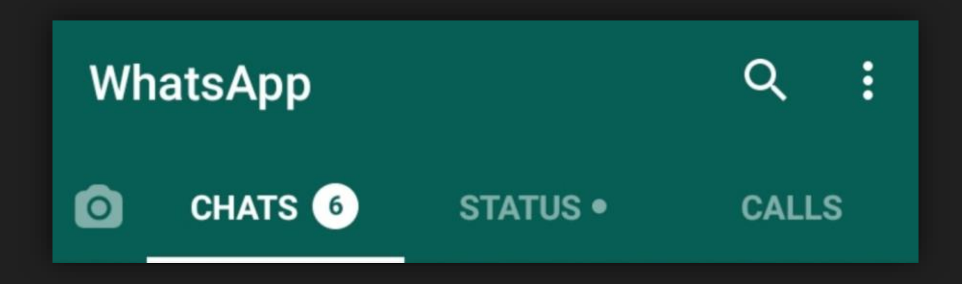


قسم Segment:

عند وجود أكثر من خيار، فوضع الخيارات ضمن أقسام أفضل من وضعها على شكل أزرار اختيار RadioButton.



يمكن أن تكون هذه الأداة أداة رئيسية في التطبيق للانتقال بين أقسامه:

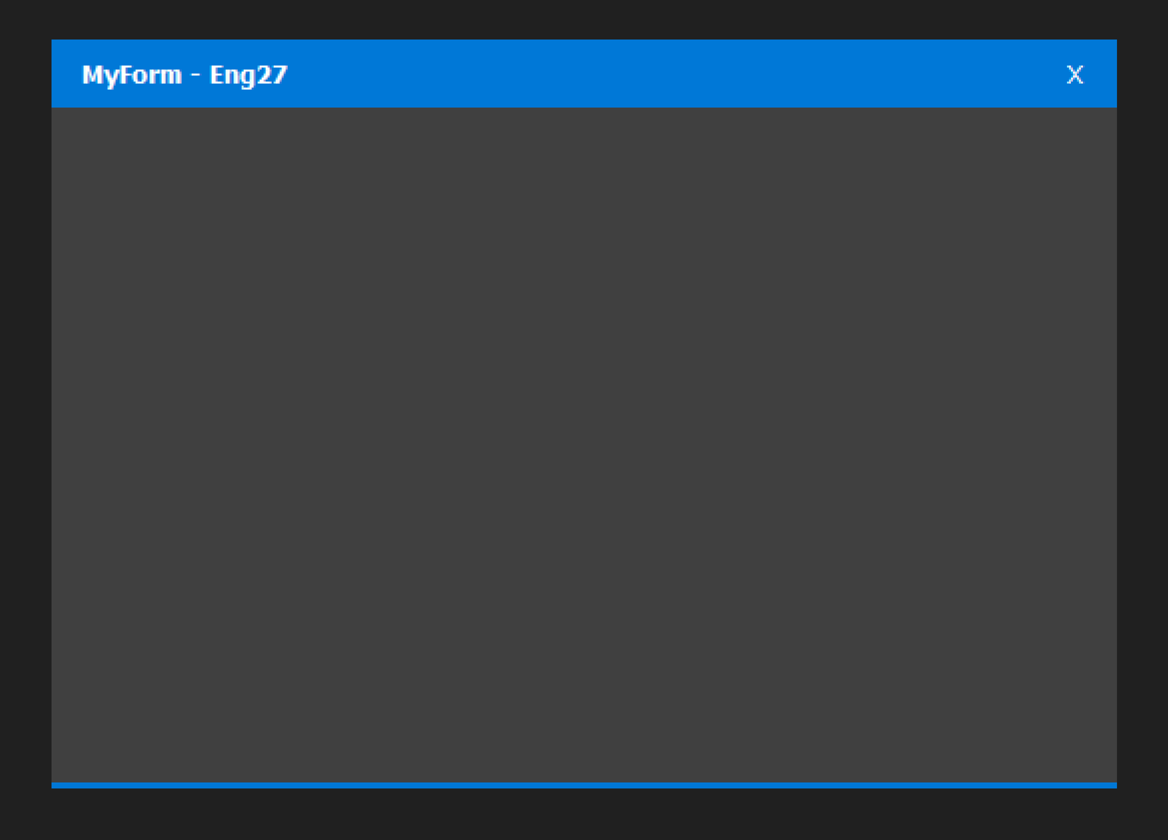


أداة تقييم RatingTool:

مجموعة من الأشكال ذات الحواف التي تظهر مفرغة أو ممتلئة بحسب قيمة التقييم، يمكن جعلها خطية أو دائرية.

نموذج خاص UserForm:

عند إنشاء نماذج Forms لا حواف لها Border Less Forms فإنك ستفقد إمكانية تحريكها، إذ إن تحريكها كان من خلال شريط العنوان.



قد تتساءل: ما الفرق بين هذا التصميم والتصميم الذي نحصل عليه من خلال النموذج التقليدي؟ والحقيقة أن الفرق كبير، يبدأ بموقع عنوان النافذة، ويمر بأزرار التحكم التي تضعها وفق حاجتك (ليس بالضرورة أن تكون ثلاثة أزرار فقط)، وينتهي بالنمط العام للنافذة (لاحظ الخط السفلي، وجوده مع شريط العنوان يضفي على النافذة طابعًا تصميميًّا).

هناك الكثير من الأمور التي يمكنك إضافتها للنوافذ – إذا ما جعلتها عديمة الحواف – وسنتناول هنا التحريك فقط، هذا الكود يعطيك إمكانية تحريك النافذة عند الضغط والسحب على أي أداة من تريد:



```
using System.Drawing;
using System.Windows.Forms;

namespace UserForm
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            pnlTitleBar.MouseDown += newMouseEventHandler(Control_MouseDown);
            pnlTitleBar.MouseMove += newMouseEventHandler(Control_MouseMove);
            lblTitleBar.MouseDown += newMouseEventHandler(Control_MouseDown);
            lblTitleBar.MouseMove += newMouseEventHandler(Control_MouseMove);
        }

        Point LastPoint;
        private void Control_MouseDown(object sender, MouseEventArgs e)
        {
            LastPoint = new Point(-e.X, -e.Y);
        }

        private void Control_MouseMove(object sender, MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
            {
                Point MousePosition = Control.MousePosition;
                MousePosition.Offset(LastPoint.X, LastPoint.Y);
                Location = MousePosition;
            }
        }
    }
}
```

ماذا عن WPF؟

محتوى هذا الكتاب موجه لمبرمجي النوافذ WinForms، وهذا الباب من الكتاب مبني على هذا الأساس. ولكن هذا لا يعني أن الطريقة المثلى لإنشاء الأدوات والتصاميم هي من خلال تطبيقات النوافذ، فالمرئيات – خصوصًا عندما يتعلق الموضوع بالحركة Animation – يمكن الحصول عليها بسهولة من خلال WPF؛ لذلك إذا كان تطبيقك يغلب عليه المرئيات أو أنه يرتكز عليها، أنشئه من خلال WPF.

وقد آثرت أن أختم هذا الفصل بهذه الفقرة ليكون انطباعك الأخير عن تصميم الأدوات أن الأصل فيه من خلال WPF.

الفصل الثامن – الفئات الرياضية

البرمجة مبنية على أسس رياضية، لكنك لا يمكنك القيام بعمليات رياضية بشكل مباشر من لغات البرمجة. صحيح أنه يمكنك القيام ببعض العمليات الرياضية الشائعة مثل التوابع المثلثية أو الجذور أو الرفع لقوة أو غيرها من العمليات، لكن لا يمكنك القيام بالتكامل أو الاشتقاق مثلًا من خلال لغات البرمجة بشكل مباشر، كما لا يمكنك أصلًا تمثيل التوابع الرياضية لتجري عليها هذه العمليات. حتى المصفوفات Arrays في البرمجة ليست المصفوفات الرياضية Matrices (مفردها مصفوفة Matrix)، فالمصفوفات البرمجية ليست إلا أشعة Vectors، حتى متعددة الأبعاد منها ما هي إلا أشعة من الأشعة (هذا يعني أن المصفوفة الثنائية في البرمجة هي شعاع من الأشعة، وقد رأينا ذلك في الفصل السابع، عند تخزين الملفات في مصفوفات).

وهذا ما يدفع كل المهندسين / المبرمجين الراغبين بالقيام بمثل هذه العمليات برمجيًّا للانتقال لبرامج هندسية مثل الماتلاب أو أوكتاف وغيرها، وهذا منطقي وسليم، لكنه مقيّد ومحدود في مواضع معينة. فكان لا بد من محاولة الوصول للتقنيات الرياضية المتقدمة والمعقدة بلغاتنا البرمجية التي نصمم وفقها برامجنا وتطبيقاتنا.

ما ستقرؤه في هذا الفصل هو كيفية التفكير بشكل كائني التوجه للوصول لهذه الأشياء الرياضية برمجيًّا. جميع فئات هذا الفصل هي ضمن مجال أسماء الفئات الرياضية Eng27.CodeBank.Math، والذي سيحوي فئات رياضية وأخرى فيزيائية.

نصيحة أخوية: إذا لم تكن تحب الرياضيات ولا الفيزياء؛ فلا تقرأ هذا الفصل.

الفئة System.Math

الفئة Math فيها كل ما تعرفه مايكروسوفت عن الرياضيات، وهي فئة ستاتيكية static (يمكنك الوصول لأعضاءها دون استنساخها). فيها ثابتين ومجموعة من التوابع. هي غير كافية إذا كنت ترغب بإجراء عمليات متقدمة في الرياضيات، لكنها تشمل معظم ما يحتاجه غير الرياضيين أو الفيزيائيين من المبرمجين في برامجهم.

أساسيات رياضية

لا أقصد بهذه الفقرة أساسيات تعليمية رياضية، وإنما أفترض أن القارئ على اطلاع جيد بالمواضيع المتقدمة في الرياضيات والفيزياء، فضلًا عن الأساسيات! وربما قد لمست هذا عند حديثنا عن رسم التوابع الرياضية في الفصل الخامس.

الثوابت Constants

على مبدأ الفئة System.Math، فإن الفئة Eng27.CodeBank.Math.Constants يجب أن تكون ستاتيكية، وبالمثل جميع أعضاءها يجب أن تكون كذلك. لا حاجة للمبرمجين لإنشاء كائنات من هذه الفئة، لذلك يجب أن تكون مغلفة sealed.

على سبيل المثال، ستحوي هذه الفئة أربعة قيم ثابتة، العدد π والعدد e، وتسارع الجاذبية الأرضية وسرعة الصوت. طور هذه الفئة بإضافة ثوابت أخرى ترى أهميتها في المكتبة.



```
namespace Eng27.CodeBank.Math
{
    ///
    public sealed class Constants
    {
        /// <summary>
        /// Represents the ratio of the circumference of a circle to its diameter,
        /// specified by the constant, π.
        /// </summary>
        public const double PI = System.Math.PI;
        /// <summary>
        /// Represents the natural logarithmic base, specified by the constant, e.
        /// </summary>
        public const double E = System.Math.E;
        /// <summary>
        /// Represents the gravitational acceleration.
        /// </summary>
        public static readonly Acceleration g = new Acceleration(9.80665);
        /// <summary>
        /// Represents the speed of light.
        /// </summary>
        public static readonly Speed C = new Speed(299792458);
    }
}
```

على اعتبار العددين π وe معرَّفَين بشكل تقريبي بالفئة Math فسنعتمد عليها، أما تسارع الجاذبية الأرضية وسرعة الضوء فعلينا تعريفها، وهي قيم فيزيائية، سنناقشها في الفقرات القادمة.

لا يمكنك التصريح عن الكائنات على أنها ثوابت، وعوضًا عن ذلك يمكنك جعلها للقراءة فقط وإسناد قيم لها لحظة التصريح عنها.



لا تحتاج الثوابت لأن تكون ستاتيكية، بينما بقية أعضاء الفئات تحتاج لذلك.

الواحدات Units

القيم الرياضية هي قيم مجردة، لا واحدة لها، على عكس القيم الفيزيائية. تتميز القيم الفيزيائية بمجموعة من الخصائص، لن نتناول منها إلا القيمة والواحدة.

واحدات القيم الفيزيائية محدودة، لذلك يمكن تمثيلها بالمعددات. مع كل فئة تمثل قيمة فيزيائية سنضع المعدد الذي يمثل واحدتها.

التعامل مع المعدّدات نصيًّا

على اعتبار أننا سنعتمد المعددات لتمثيل الواحدات، فقد تحتاج لإدخالها أو الحصول عليها كقيم نصية، صحيح أننا لن نعتمد على هذا الأسلوب في الفئات القادمة وإنما سندخل الواحدات بشكل مباشر من خلال المعددات، إلا أني سأضع طريقتين قد تحتاجهما مستقبلًا.



```
using System;

namespace Eng27.CodeBank
{
    ///
    public static class Enums
    {
        ///
        public static T StringToEnum<T>(string str)
        {
            T t;
            t = (T)Enum.Parse(typeof(T), str);
            return t;
        }

        ///
        public static string[] EnumToArray<T>()
        {
            return Enum.GetNames(typeof(T));
        }
    }
}
```

الطريقة StringToEnum تأخذ وسيطًا نصيًّا وتعيد معدّدًا من النوع T (نوع غير محدد يحدّده المبرمج بين قوسين < > عند استدعاءها، أما الطريقة EnumToArray فلا تأخذ

وسطاء ولكنها تعيد جميع عناصر المعدد كمصفوفة نصية، بعد تحديد المعدّد كما في الطريقة الأولى.

يمكنك تطوير هذه الفئة وجعل طرقها ضمن الفئة ExtensionMethods، لكن مبدئيًّا الشكل الحالي لهذه الطرق جيد.

الفئة MathObject

جميع الفئات الرياضية – والفيزيائية – التي سننشئها ستشترك بأعضاء عامة، هذه الأعضاء هي على سبيل المثال: دقة الكائن الرياضي، وحدث يحدث عند تغيير الدقة. (نقصد بالدقة: عدد الأرقام بعد الفاصلة)



```
using System;

namespace Eng27.CodeBank.Math
{
    ///
    public class MathObject
    {
        private int precision = 3;
        ///
        public int Precision
        {
            get { return precision; }
            set
            {
                if (value < 0 || value > 9)
                    throw new PrecisionException(value);
                precision = value;
                this.OnPrecisionChange(new EventArgs());
            }
        }

        ///
        public event EventHandler PrecisionChange;
        ///
        protected virtual void OnPrecisionChange(EventArgs e)
        {
            if (PrecisionChange != null)
                PrecisionChange.Invoke(this, e);
        }
    }
}
```

عند إدخال رقم لا يصلح لأن يمثل دقة العدد، يحدث الاستثناء PrecisionException:



```
using System;

namespace Eng27.CodeBank.Math
{
    public class PrecisionException : Exception
    {
        string msg = "";
        ///
        public PrecisionException(int val)
            : base("Precision Must be positive and less than 10.")
        {
            this.Value = val;
            if (val < 0)
                msg = "Precision Must be positive.";
            else if (val > 9)
                msg = "Precision Must not be more than 9.";
        }

        ///
        public int Value { get; private set; }

        ///
        public override string Message
        {
            get
            {
                return this.msg;
            }
        }
    }
}
```

الفئة PhysicalObject

الكائنات الفيزيائية هي في الأساس كائنات عددية (MathObject) إلا أن فيها أعضاء إضافية تجعلها حقيقية (ليست مجردة كالكائنات الرياضية).



```
namespace Eng27.CodeBank.Math
{
    ///
    public abstract class PhysicalObject : MathObject
    {
        private double val;
        ///
        public double Value
        {
            get { return val; }
            set { val = value; }
        }
```

```
        ///
        public abstract string UnitToString();
    }
}
```

الطريقة UnitToString لا نعرف محتواها، لأنها تختلف من كائن فيزيائي لغيره (تتعلق بواحدات كل كائن)، لكننا نحتاج وجودها في جميع الكائنات الفيزيائية، لذلك سنجعلها مجردة؛ إذ إن الطرق المجردة تجبر جميع الفئات التي سترث من الفئة الموجودة فيها على توظيفها Implementing.

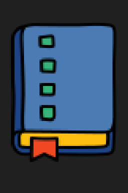
واحدات الكائن الفيزيائي أيضًا يجب أن تتواجد في جميع الكائنات الفيزيائية، لكننا لا نعرف نوعها، على عكس الطريقة UnitToString فنحن نعرف نوع القيمة التي ستعيدها (قيمة نصية)، إذ إنها معددات وليست فئات، ولا يمكن الوراثة من المعددات. لذلك سنترك الواحدات لكل فئة بفئتها.

العمليات الجبرية، برمجيًّا

العمليات الجبرية الأساسية أربعة: الجمع والطرح والضرب والقسمة، وهي ليست حكرًا على المتغيرات العددية int وdouble وغيرها، فحتى المتغيرات النصية يمكنها تطبيق عملية الجمع وعندها لا يكون الناتج قيمة رياضية، بل قيمة نصية. وهي أيضًا ليست حكرًا على مايكروسوفت، فحتى فئاتك الخاصة يمكنها تطبيق العمليات الجبرية، ونوع القيمة الناتجة من هذه العمليات أنت من تحدده، كما أن العمليات الجبرية هنا لا تتضمن فقط العمليات الأربعة، وإنما يمكنك التصريح عن عمليات أخرى كالرفع إلى قوة باستخدام الرمز ^.

يمكن تعريف العمليات الجبرية للفئات من خلال الكلمة operator، وبعدها رمز العملية، وبعدها بين قوسين كائنين: أحدهما أو كلاهما الفئة نفسها. أما نوع الكائن الذي سيتم إرجاعه فأنت من تحدده.

يمكنك تعريف عملية جبرية – لعملية الجمع مثلًا – لفئة من فئاتك بالصيغة التالية:



```
public static ClassType operator +(ClassType1 obj1, ClassType2 obj2)
{
    //.
    //.
    //Codes

    ClassType class_type = new ClassType();
    return class_type;
}
```

فئات فيزيائية

على سبيل نمطية الفئات، جميع الفئات في هذه الفقرة يمكن إنشاء كائنات منها دون تمرير قيمة أو واحدة إليها، أو بتمرير قيمة فقط (لتعرَّف وفق الواحدة القياسية)، أو بتمرير قيمة وواحدة.

كما توجد فيها كلها طريقتين ConvertToSI وConvertTo، الأولى تحوّل الكائن لكائن من نفس النوع لكن بالواحدة القياسية، والثانية تحوّله لكائن بواحدة معينة. في الواقع فإن هاتين الطريقتين تعيدان كائنًا جديدًا بإعادة استنساخ نفس الفئة.

عند استدعاء الطريقة ToString لهذه الكائنات، فإنه يتم إرجاع قيمة الكائن ونوعه بعد تحويله لقيمة نصية (باستدعاء الطريقة UnitToString)، فالسرعة التي قيمتها 1 وواحدتها m/s ترجع القيمة النصية "1 m/s" عند استدعاء الطريقة ToString.

أما العمليات الجبرية على هذه الكائنات فتعيد إما كائنًا من النوع نفسه أو كائنًا من نوع آخر بحسب أطراف العملية الجبرية. فعملية قسمة السرعة على الزمن يعطي تسارعًا.

يجب تحويل أطراف العملية الجبرية للواحدات القياسية للحصول على كائنات بواحدات قياسية، لذلك فإنك سترى بعض التحويلات ضمن التوابع التي تعرّف العمليات الجبرية لهذه الكائنات. وعلى سبيل المثال: عند قسمة مسافة $1\ cm$ على زمن $1\ min$ فإن الناتج يجب أن يكون سرعة $1\ cm/min$، لكن هذا سيدخلنا في معضلة تحديد واحدات أطراف العمليات الجبرية بشروط عديدة ومتداخلة، لذلك فالنتيجة التالية أسهل:

$$\frac{1\ cm}{1\ min} = \frac{0.01\ m}{60\ s} = 0.000167\ m/s$$

وبعدها، يمكن للمبرمج تحويل الناتج للواحدة التي يرغب بها من خلال الطريقة ConvertTo بتمرير معدد يحدد الواحدة المطلوبة.

وأخيرًا، لا بد من تغيير دقة قيمة الكائن في كل مرة يحدث فيها الحدث PrecisionChange.

المسافة Distance



```
using System;

namespace Eng27.CodeBank.Math
{
    ///
    public class Distance : PhysicalObject
    {
        #region Constructors
        ///
        public Distance()
        {
            this.PrecisionChange += Distance_PrecisionChange;
        }

        ///
        public Distance(double Value, DistanceUnits_e unit)
        {
            this.Value = Value;
            d_unit = unit;
            this.PrecisionChange += Distance_PrecisionChange;
        }

        ///
        public Distance(double Value)
        {
            this.Value = Value;
            d_unit = DistanceUnits_e.Meter;
            this.PrecisionChange += Distance_PrecisionChange;
        }
        #endregion


        #region Properties
        private DistanceUnits_e d_unit;
        ///
        public DistanceUnits_e Unit
        {
            get { return d_unit; }
            set { d_unit = value; }
        }
        #endregion
```

```
        #region Methods
        ///
        public Distance ConvertToSI()
        {
            double dist = this.Value;
            switch (d_unit)
            {
                case DistanceUnits_e.Millimeter:
                    dist *= 1e-3;
                    break;
                case DistanceUnits_e.Centimeter:
                    dist *= 1e-2;
                    break;
                case DistanceUnits_e.Meter:
                    break;
                case DistanceUnits_e.Kilometer:
                    dist *= 1e3;
                    break;
                case DistanceUnits_e.Inch:
                    dist *= 2.54e-2;
                    break;
                case DistanceUnits_e.Foot:
                    dist *= 3.048e-1;
                    break;
                case DistanceUnits_e.Yard:
                    dist *= 9.144e-1;
                    break;
                case DistanceUnits_e.Mile:
                    dist *= 1.6093e3;
                    break;
                default:
                    break;
            }

            return new Distance(dist);
        }

        ///
        public Distance ConvertTo(DistanceUnits_e dist_unit)
        {
            double ans = this.Value;
            if (d_unit == dist_unit)
                goto end;

            Distance a = this.ConvertToSI();
            ans = a.Value;
            //unit is m/s
            switch (dist_unit)
            {
                case DistanceUnits_e.Millimeter:
                    ans *= 1e3;
                    break;
                case DistanceUnits_e.Centimeter:
                    ans *= 1e2;
                    break;
                case DistanceUnits_e.Meter:
                    break;
                case DistanceUnits_e.Kilometer:
                    ans *= 1e-3;
                    break;
```

```
            case DistanceUnits_e.Inch:
                ans *= 3.937e1;
                break;
            case DistanceUnits_e.Foot:
                ans *= 3.2808;
                break;
            case DistanceUnits_e.Yard:
                ans *= 1.0936;
                break;
            case DistanceUnits_e.Mile:
                ans *= 6.2137e-4;
                break;
            default:
                break;
        }
    end:
        return new Distance(ans, dist_unit);
    }

    ///
    public override string UnitToString()
    {
        string u = "";
        switch (d_unit)
        {
            case DistanceUnits_e.Millimeter:
                u = "mm";
                break;
            case DistanceUnits_e.Centimeter:
                u = "cm";
                break;
            case DistanceUnits_e.Meter:
                u = "m";
                break;
            case DistanceUnits_e.Kilometer:
                u = "km";
                break;
            case DistanceUnits_e.Inch:
                u = "Inch";
                break;
            case DistanceUnits_e.Foot:
                u = "ft";
                break;
            case DistanceUnits_e.Yard:
                u = "yd";
                break;
            case DistanceUnits_e.Mile:
                u = "Mile";
                break;
            default:
                break;
        }
        return u;
    }

    ///
    public override string ToString() {
        return this.Value + " " + UnitToString();
    }
    #endregion
```

```
        #region Operators
        public static Distance operator +(Distance dist1, Distance dist2)
        {
            // يجب تحويل الأعداد للواحدات القياسية
            dist1 = dist1.ConvertToSI();
            dist2 = dist2.ConvertToSI();
            // ثم إرجاع كائن بواحدة القياس القياسية
            return new Distance(dist1.Value + dist2.Value, DistanceUnits_e.Meter);
        }

        public static Distance operator -(Distance dist1, Distance dist2)
        {
            dist1 = dist1.ConvertToSI();
            dist2 = dist2.ConvertToSI();
            return new Distance(dist1.Value - dist2.Value, DistanceUnits_e.Meter);
        }

        public static Distance operator *(Distance dist, double n)
        {
            return new Distance(dist.Value * n, dist.Unit);
        }

        public static Distance operator *(double n, Distance dist)
        {
            return new Distance(dist.Value * n, dist.Unit);
        }

        public static Distance operator /(Distance dist, double n)
        {
            return new Distance(dist.Value / n, dist.Unit);
        }

        public static double operator /(Distance dist1, Distance dist2)
        {
            // عند قسمة قيمة فيزيائية على نفسها (أو على قيمة فيزيائية أخرى من نفس النوع) فلا واحدة قياس للناتج
            dist1 = dist1.ConvertToSI();
            dist2 = dist2.ConvertToSI();
            return new Distance(dist1.Value / dist2.Value,
                DistanceUnits_e.Meter).Value;
        }

        public static Speed operator /(Distance dist, Time time)
        {
            dist = dist.ConvertToSI();
            time = time.ConvertToSI();
            return new Speed(dist.Value / time.Value,
                SpeedUnits_e.MeterPerSecond);
        }
        #endregion


        #region Events
        void Distance_PrecisionChange(object sender, EventArgs e)
        {
            this.Value = System.Math.Round(this.Value, this.Precision);
        }
        #endregion
    }
}
```

واحدات المسافة معرفة بالمعدد DistanceUnits_e:



```
namespace Eng27.CodeBank.Math
{
    ///
    public enum DistanceUnits_e
    {
        Millimeter,
        Centimeter,
        Meter,
        Kilometer,
        Inch,
        Foot,
        Yard,
        Mile,
    }
}
```

الزمن Time



```
using System;

namespace Eng27.CodeBank.Math
{
    ///
    public class Time : PhysicalObject
    {
        #region Constructors
        ///
        public Time()
        {
            this.PrecisionChange += Time_PrecisionChange;
        }

        ///
        public Time(double Value, TimeUnits_e unit)
        {
            this.Value = Value;
            t_unit = unit;
            this.PrecisionChange += Time_PrecisionChange;
        }

        ///
        public Time(double Value)
        {
            this.Value = Value;
            t_unit = TimeUnits_e.Second;
            this.PrecisionChange += Time_PrecisionChange;
        }
        #endregion
```

```
        #region Properties
        private TimeUnits_e t_unit;
        ///
        public TimeUnits_e Unit
        {
            get { return t_unit; }
            set { t_unit = value; }
        }
        #endregion


        #region Methods
        ///
        public Time ConvertToSI()
        {
            double time = this.Value;
            switch (this.Unit)
            {
                case TimeUnits_e.Second:
                    break;
                case TimeUnits_e.MilliSecond:
                    time *= 1e-3;
                    break;
                case TimeUnits_e.MicroSecond:
                    time *= 1e-6;
                    break;
                case TimeUnits_e.Minute:
                    time *= 60;
                    break;
                case TimeUnits_e.Hour:
                    time *= 3600;
                    break;
                case TimeUnits_e.Day:
                    time *= 8.64e4;
                    break;
                case TimeUnits_e.Week:
                    time *= 6.048e5;
                    break;
                case TimeUnits_e.Month:
                    time *= 2.628e6;
                    break;
                case TimeUnits_e.Year:
                    time *= 3.1536e7;
                    break;
                default:
                    break;
            }

            return new Time(time);
        }

        ///
        public Time ConvertTo(TimeUnits_e time_unit)
        {
            double ans = this.Value;
            if (t_unit == time_unit)
                goto end;
```

```
        // الواحدة قد لا تكون ثا
        Time t = this.ConvertToSI();
        ans = t.Value;
        // الآن، الواحدة هي ثا
        switch (time_unit)
        {
            case TimeUnits_e.Second:
                break;
            case TimeUnits_e.MilliSecond:
                ans *= 1e3;
                break;
            case TimeUnits_e.MicroSecond:
                ans *= 1e6;
                break;
            case TimeUnits_e.Minute:
                ans /= 60;
                break;
            case TimeUnits_e.Hour:
                ans /= 3600;
                break;
            case TimeUnits_e.Day:
                ans *= 1.1574e-5;
                break;
            case TimeUnits_e.Week:
                ans *= 1.6534e-6;
                break;
            case TimeUnits_e.Month:
                ans *= 3.8052e-7;
                break;
            case TimeUnits_e.Year:
                ans *= 3.171e-8;
                break;
            default:
                break;
        }
    end:
        return new Time(ans, time_unit);
    }

    ///
    public override string UnitToString()
    {
        string u = "";
        switch (t_unit)
        {
            case TimeUnits_e.Second:
                u = "s";
                break;
            case TimeUnits_e.MilliSecond:
                u = "ms";
                break;
            case TimeUnits_e.MicroSecond:
                u = "\u03bcs";
                break;
            case TimeUnits_e.Minute:
                u = "min";
                break;
            case TimeUnits_e.Hour:
                u = "hr";
```

```
                    break;
                case TimeUnits_e.Day:
                    u = "day";
                    break;
                case TimeUnits_e.Week:
                    u = "week";
                    break;
                case TimeUnits_e.Month:
                    u = "month";
                    break;
                case TimeUnits_e.Year:
                    u = "year";
                    break;
                default:
                    break;
            }
            return u;
        }

        ///
        public override string ToString()
        {
            return this.Value + " " + UnitToString();
        }
        #endregion


        #region Events
        ///
        private void Time_PrecisionChange(object sender, EventArgs e)
        {
            this.Value = System.Math.Round(this.Value, this.Precision);
        }
        #endregion
    }
}
```

واحدات الزمن:



```
namespace Eng27.CodeBank.Math
{
    ///
    public enum TimeUnits_e
    {
        Second,
        MilliSecond,
        MicroSecond,
        Minute,
        Hour,
        Day,
        Week,
        Month,
        Year
    }
}
```

السرعة Speed



```
namespace Eng27.CodeBank.Math
{
    ///
    public class Speed : PhysicalObject
    {
        #region Constructors
        ///
        public Speed()
        {
            this.PrecisionChange += Speed_PrecisionChange;
        }

        ///
        public Speed(double Value, SpeedUnits_e Unit)
        {
            this.Value = Value;
            s_unit = Unit;
            this.PrecisionChange += Speed_PrecisionChange;
        }

        ///
        public Speed(double Value)
        {
            this.Value = Value;
            s_unit = SpeedUnits_e.MeterPerSecond;
            this.PrecisionChange += Speed_PrecisionChange;
        }
        #endregion


        #region Properties
        private SpeedUnits_e s_unit;
        ///
        public SpeedUnits_e Unit
        {
            get { return s_unit; }
            set { s_unit = value; }
        }
        #endregion


        #region Methods
        ///
        public Speed ConvertToSI()
        {
            double speed = this.Value;
            switch (this.Unit)
            {
                case SpeedUnits_e.MillimeterPerSecond:
                    speed /= 1e3;
                    break;
                case SpeedUnits_e.CenitimeterPerSecond:
                    speed /= 1e2;
                    break;
                case SpeedUnits_e.MeterPerSecond:
                    break;
```

```
            case SpeedUnits_e.KilometerPerHour:
                speed *= 0.27778;
                break;
            case SpeedUnits_e.MilePerHour:
                speed *= 0.44704;
                break;
            case SpeedUnits_e.Mach:
                speed *= 295.05;
                break;
            default:
                break;
        }
        return new Speed(speed);
    }

    ///
    public Speed ConvertTo(SpeedUnits_e speed_unit)
    {
        double ans = this.Value;
        if (s_unit == speed_unit)
            goto end;

        Speed s = this.ConvertToSI();
        ans = s.Value;

        switch (speed_unit)
        {
            case SpeedUnits_e.MillimeterPerSecond:
                ans *= 1e3;
                break;
            case SpeedUnits_e.CenitimeterPerSecond:
                ans *= 1e2;
                break;
            case SpeedUnits_e.MeterPerSecond:
                break;
            case SpeedUnits_e.KilometerPerHour:
                ans /= 3.6;
                break;
            case SpeedUnits_e.MilePerHour:
                ans /= 2.2369;
                break;
            case SpeedUnits_e.Mach:
                ans /= 3.3893e-3;
                break;
            default:
                break;
        }
    end:
        return new Speed(ans, speed_unit);
    }

    ///
    public override string UnitToString()
    {
        string u = "";
        switch (s_unit)
        {
            case SpeedUnits_e.MillimeterPerSecond:
                u = "mm/s";
                break;
```

```
                case SpeedUnits_e.CenitimeterPerSecond:
                    u = "cm/s";
                    break;
                case SpeedUnits_e.MeterPerSecond:
                    u = "m/s";
                    break;
                case SpeedUnits_e.KilometerPerHour:
                    u = "km/h";
                    break;
                case SpeedUnits_e.MilePerHour:
                    u = "M/h";
                    break;
                case SpeedUnits_e.Mach:
                    u = "Mach";
                    break;
                default:
                    break;
            }
            return u;
        }

        ///
        public override string ToString()
        {
            return this.Value + " " + UnitToString();
        }
        #endregion


        #region Operators
        public static Speed operator +(Speed speed1, Speed speed2)
        {
            speed1 = speed1.ConvertToSI();
            speed2 = speed2.ConvertToSI();
            return new Speed(speed1.Value + speed2.Value,
                SpeedUnits_e.MeterPerSecond);
        }

        public static Speed operator -(Speed speed1, Speed speed2)
        {
            speed1 = speed1.ConvertToSI();
            speed2 = speed2.ConvertToSI();
            return new Speed(speed1.Value - speed2.Value,
                SpeedUnits_e.MeterPerSecond);
        }

        public static Speed operator *(Speed speed, double n)
        {
            return new Speed(speed.Value * n, speed.Unit);
        }

        public static Speed operator *(double n, Speed speed)
        {
            return new Speed(speed.Value * n, speed.Unit);
        }

        public static Speed operator /(Speed speed, double n)
        {
            return new Speed(speed.Value / n, speed.Unit);
        }
```

```
        public static double operator /(Speed speed1, Speed speed2)
        {
            speed1 = speed1.ConvertToSI();
            speed2 = speed2.ConvertToSI();
            return new Speed(speed1.Value / speed2.Value,
                SpeedUnits_e.MeterPerSecond).Value;
        }

        public static Acceleration operator /(Speed speed, Time time)
        {
            speed = speed.ConvertToSI();
            time = time.ConvertToSI();
            return new Acceleration(speed.Value / time.Value,
                AccelerationUnits_e.MeterPerSecond2);
        }

        public static Distance operator *(Speed speed, Time time)
        {
            speed = speed.ConvertToSI();
            time = time.ConvertToSI();
            return new Distance(speed.Value * time.Value, DistanceUnits_e.Meter);
        }

        public static Distance operator *(Time time, Speed speed)
        {
            speed = speed.ConvertToSI();
            time = time.ConvertToSI();
            return new Distance(speed.Value * time.Value, DistanceUnits_e.Meter);
        }
        #endregion


        #region Events
        ///
        void Speed_PrecisionChange(object sender, System.EventArgs e)
        {
            this.Value = System.Math.Round(this.Value, this.Precision);
        }
        #endregion
    }
}
```

واحدات السرعة:



```
namespace Eng27.CodeBank.Math
{
    ///
    public enum SpeedUnits_e
    {
        MillimeterPerSecond,
        CenitimeterPerSecond,
        MeterPerSecond,
        KilometerPerHour,
        MilePerHour,
        Mach
    }
}
```

التسارع Acceleration



```
using System;

namespace Eng27.CodeBank.Math
{
    public class Acceleration : PhysicalObject {
        #region Constructors
        ///
        public Acceleration()
        {
            this.PrecisionChange += Acceleration_PrecisionChange;
        }

        ///
        public Acceleration(double Value, AccelerationUnits_e unit)
        {
            this.Value = Value;
            a_unit = unit;
            this.PrecisionChange += Acceleration_PrecisionChange;
        }

        ///
        public Acceleration(double Value)
        {
            this.Value = Value;
            a_unit = AccelerationUnits_e.MeterPerSecond2;
            this.PrecisionChange += Acceleration_PrecisionChange;
        }
        #endregion


        #region Properties
        private AccelerationUnits_e a_unit;
        ///
        public AccelerationUnits_e Unit
        {
            get { return a_unit; }
            set { a_unit = value; }
        }
        #endregion


        #region Methods
        ///
        public Acceleration ConvertToSI() {
            double acc = this.Value;
            switch (this.Unit)
            {
                case AccelerationUnits_e.MillimeterPerSecond2:
                    acc *= 1e-3;
                    break;
                case AccelerationUnits_e.CentimeterPerSecond2:
                    acc *= 1e-2;
                    break;
                case AccelerationUnits_e.MeterPerSecond2:
                    break;
                case AccelerationUnits_e.KilometerPerSecond2:
```

```
                acc *= 1e3;
                break;
            case AccelerationUnits_e.InchPerSecond2:
                acc *= 2.54e-2;
                break;
            case AccelerationUnits_e.FootPerSecond2:
                acc *= 3.048e-1;
                break;
            case AccelerationUnits_e.YardPerSecond2:
                acc *= 9.144e-1;
                break;
            case AccelerationUnits_e.MilePerSecond2:
                acc *= 1.6093e3;
                break;
            default:
                break;
        }
        return new Acceleration(acc);
    }

    ///
    public Acceleration ConvertTo(AccelerationUnits_e acc_unit) {
        double ans = this.Value;
        if (a_unit == acc_unit)
            goto end;

        Acceleration a = this.ConvertToSI();
        ans = a.Value;

        switch (acc_unit)
        {
            case AccelerationUnits_e.MillimeterPerSecond2:
                ans *= 1e3;
                break;
            case AccelerationUnits_e.CentimeterPerSecond2:
                ans *= 1e2;
                break;
            case AccelerationUnits_e.MeterPerSecond2:
                break;
            case AccelerationUnits_e.KilometerPerSecond2:
                ans *= 1e-3;
                break;
            case AccelerationUnits_e.InchPerSecond2:
                ans *= 3.937e1;
                break;
            case AccelerationUnits_e.FootPerSecond2:
                ans *= 3.2808;
                break;
            case AccelerationUnits_e.YardPerSecond2:
                ans *= 1.0936;
                break;
            case AccelerationUnits_e.MilePerSecond2:
                ans *= 6.2137e-4;
                break;
            default:
                break;
        }
    end:
        return new Acceleration(ans, acc_unit);
    }
```

```
        ///
        public override string UnitToString()
        {
            string u = "";
            switch (a_unit)
            {
                case AccelerationUnits_e.MillimeterPerSecond2:
                    u = "mm/s2";
                    break;
                case AccelerationUnits_e.CentimeterPerSecond2:
                    u = "cm/s2";
                    break;
                case AccelerationUnits_e.MeterPerSecond2:
                    u = "m/s2";
                    break;
                case AccelerationUnits_e.KilometerPerSecond2:
                    u = "km/s2";
                    break;
                case AccelerationUnits_e.InchPerSecond2:
                    u = "In/s2";
                    break;
                case AccelerationUnits_e.FootPerSecond2:
                    u = "ft/s2";
                    break;
                case AccelerationUnits_e.YardPerSecond2:
                    u = "yd/s2";
                    break;
                case AccelerationUnits_e.MilePerSecond2:
                    u = "M/s2";
                    break;
                default:
                    break;
            }
            return u;
        }

        ///
        public override string ToString()
        {
            return this.Value + " " + UnitToString();
        }
        #endregion


        #region Operators
        public static Acceleration operator +(Acceleration acc1,Acceleration acc2)
        {
            acc1 = acc1.ConvertToSI();
            acc2 = acc2.ConvertToSI();
            return new Acceleration(acc1.Value + acc2.Value,
                AccelerationUnits_e.MeterPerSecond2);
        }

        public static Acceleration operator -(Acceleration acc1,Acceleration acc2)
        {
            acc1 = acc1.ConvertToSI();
            acc2 = acc2.ConvertToSI();
            return new Acceleration(acc1.Value - acc2.Value,
                AccelerationUnits_e.MeterPerSecond2);
        }
```

```
        public static Acceleration operator *(Acceleration acc, double n)
        {
            return new Acceleration(acc.Value * n, acc.Unit);
        }

        public static Acceleration operator *(double n, Acceleration acc)
        {
            return new Acceleration(acc.Value * n, acc.Unit);
        }

        public static Acceleration operator /(Acceleration acc, double n)
        {
            return new Acceleration(acc.Value / n, acc.Unit);
        }

        public static double operator /(Acceleration acc1, Acceleration acc2)
        {
            acc1 = acc1.ConvertToSI();
            acc2 = acc2.ConvertToSI();
            return new Acceleration(acc1.Value / acc2.Value,
                AccelerationUnits_e.MeterPerSecond2).Value;
        }

        public static Speed operator *(Acceleration acc, Time time)
        {
            acc = acc.ConvertToSI();
            time = time.ConvertToSI();
            return new Speed(acc.Value * time.Value, SpeedUnits_e.MeterPerSecond);
        }
        #endregion

        #region Events
        void Acceleration_PrecisionChange(object sender, EventArgs e)
        {
            this.Value = System.Math.Round(this.Value, this.Precision);
        }
        #endregion
    }
}
```

```
namespace Eng27.CodeBank.Math
{
    ///
    public enum AccelerationUnits_e
    {
        MillimeterPerSecond2,
        CentimeterPerSecond2,
        MeterPerSecond2,
        KilometerPerSecond2,
        InchPerSecond2,
        FootPerSecond2,
        YardPerSecond2,
        MilePerSecond2,
    }
}
```

وكمثال على ما سبق، جرب الكود التالي في مشروع نوافذ:

أنشئ كائنًا يمثل الزمن وأعطه القيمة 15 min، وكائنًا يمثل المسافة وأعطه القيمة 140 (على اعتبار لم يتم إدخال الواحدة فإن واحدة الكائن هي الواحدة القياسية، المتر)، ثم اجعل واحدة الكائن الكيلومتر، ثم أنشئ كائنًا يمثل السرعة وليكن ناتج قسمة كائن المسافة على كائن الزمن، وبالمثل كائن التسارع، ناتج قسمة كائن السرعة على كائن الزمن. ثم أظهر النتيجة بعد تحويلها لواحدة cm/s^2.

```
private void btnCalculation_Click(object sender, EventArgs e)
{
    Time t = new Time(15, TimeUnits_e.Minute);
    Distance d = new Distance(140);
    d.Unit = DistanceUnits_e.Kilometer;
    Speed s = d / t;
    Acceleration a = s / t;
    a.Precision = 3;
    MessageBox.Show(a.ConvertTo(AccelerationUnits_e.CentimeterPerSecond2).ToString());
}
```

17.3 cm/s2

OK

لطالما كنت أبحث عن بيئة تطوير IDE تفهم السياق فيزيائيًّا، تمامًا كالفيجوال ستوديو مع الفئات وأعضاءها، والماتلاب مع المكونات التحكمية والرياضية والرسومية، ولعلّ الصورة السابقة تحمل في طياتها شيئا من مرادي.

الأعداد العقدية

توجد الأعداد العقدية إما بالشكل الديكارتي $z = x + i\,y$ أو بالشكل القطبي (ويسمى المثلثي) $z = [R, \theta°]$، وتربط بينهما مجموعة من المعادلات الرياضية.

$$R = \sqrt{x^2 + y^2} \qquad x = R\,\cos\theta$$
$$\theta = \tan^{-1}\frac{x}{y} \qquad y = R\,\sin\theta$$

القيم x وy وR هي قيم عددية، يمكننا اعتبارها متغيرات من النوع double، أما الزاوية θ سنأخذها بالدرجات والراديان، لذلك علينا تأليف نوع جديد يمكنه احتواء الزوايا، هذا النوع سنسميه Angle، والذي يحوي خاصية تحدد واحدة قياس الزاوية (درجة أو راديان).

أيًّا كان العدد العقدي – ديكارتيًّا أو قطبيًّا – فإن الفئة ComplexNumber تمثله:



```
namespace Eng27.CodeBank.Math
{
    ///
    public abstract class ComplexNumber : MathObject
    {
        private AngleType_e angle_type;
        ///
        public AngleType_e AngleType
        {
            get { return angle_type; }
            set { angle_type = value; }
        }

        public abstract void RefreshNumber();
    }
}
```

الزاوية Angle



```
namespace Eng27.CodeBank.Math
{
    ///
    public class Angle:PhysicalObject {
        #region Constructors
        ///
        public Angle()
        {
            this.PrecisionChange += Angle_PrecisionChange;
        }

        ///
        public Angle(double Value, AngleType_e Type)
        {
            this.Value = Value;
            angle_type = Type;
            this.PrecisionChange += Angle_PrecisionChange;
        }

        ///
        public Angle(double Value)
        {
            this.Value = Value;
            angle_type = AngleType_e.Degrees;
            this.PrecisionChange += Angle_PrecisionChange;
        }
        #endregion
```

```
        #region Properties
        private AngleType_e angle_type;
        ///
        public AngleType_e AngleType
        {
            get { return angle_type; }
            set { angle_type = value; }
        }
        #endregion


        #region Methods
        ///
        public double ConvertTo(AngleType_e type)
        {
            double ans = this.Value;
            if (angle_type == type)
                goto end;

            switch (type)
            {
                case AngleType_e.Degrees:
                    ans *= 180 / Constants.PI;
                    break;
                case AngleType_e.Radians:
                    ans *= Constants.PI / 180;
                    break;
                default:
                    break;
            }
        end:
            return ans;
        }

        ///
        public override string UnitToString()
        {
            string u = "";
            switch (angle_type)
            {
                case AngleType_e.Degrees:
                    u = "\u00b0";
                    break;
                case AngleType_e.Radians:
                    u = "rad";
                    break;
                default:
                    break;
            }
            return u;
        }

        ///
        public override string ToString()
        {
            return this.Value + " " + UnitToString();
        }
        #endregion
```

```
        #region Operators
        ///
        public static Angle operator +(Angle a1, Angle a2)
        {
            a1.Value = a1.ConvertTo(AngleType_e.Degrees);
            a2.Value = a2.ConvertTo(AngleType_e.Degrees);
            return new Angle(a1.Value + a2.Value, AngleType_e.Degrees);
        }

        ///
        public static Angle operator -(Angle a1, Angle a2)
        {
            a1.Value = a1.ConvertTo(AngleType_e.Degrees);
            a2.Value = a2.ConvertTo(AngleType_e.Degrees);
            return new Angle(a1.Value - a2.Value, AngleType_e.Degrees);
        }
        #endregion


        #region Events
        ///
        void Angle_PrecisionChange(object sender, System.EventArgs e)
        {
            this.Value = System.Math.Round(this.Value, this.Precision);
        }
        #endregion
    }
}
```

أما واحدة قياس الزوايا AngleUnits_e:



```
namespace Eng27.CodeBank.Math
{
    ///
    public enum AngleType_e
    {
        Degrees,
        Radians
    }
}
```

العدد العقدي الديكارتي

إذا كان العدد العقدي بالشكل الديكارتي (يسمى أيضًا العدد العقدي بالساحة الديكارتية) فإنه يتكون من مركبتين، أو يمكننا أن نقول تجاوزًا إحداثيّين، x وy، يسميان أيضًا العدد الحقيقي والعدد التخيلي.

للعدد الديكارتي ما يسمى بالمرافق Conjugation، ويرمز له $\bar{Z}$ وهو نفسه العدد الديكارتي لكن عدده التخيلي معاكس للعدد التخيلي للعدد الديكارتي الأساسي، أي:

$$\bar{Z} = x - i\, y$$



```
namespace Eng27.CodeBank.Math
{
    ///
    public class RectangularComplexNumber : ComplexNumber
    {
        #region Constructors
        ///
        public RectangularComplexNumber()
        {
        }

        ///
        public RectangularComplexNumber(double RealNumber, double ImaginaryNumber)
        {
            real_number = RealNumber;
            imaginary_number = ImaginaryNumber;
            this.RefreshNumber();
        }
        #endregion


        #region Properties
        private double real_number;
        ///
        public double RealNumber
        {
            get { return real_number; }
            set
            {
                real_number = value;
                this.RefreshNumber();
            }
        }

        private double imaginary_number;
        ///
        public double ImaginaryNumber
        {
            get { return imaginary_number; }
            set
            {
                imaginary_number = value;
                this.RefreshNumber();
            }
        }
```

```
        ///
        public RectangularComplexNumber Conjugation
        {
            get
            {
                if (real_number != null && imaginary_number != null)
                    return new RectangularComplexNumber
                        (real_number, -imaginary_number);
                else
                    return null;
            }
        }
        #endregion


        #region Methods
        ///
        public PolarComplexNumber ToPolar()
        {
            double modulus = System.Math.Sqrt(
                real_number *
                real_number +
                imaginary_number *
                imaginary_number);
            Angle angle = new Angle();
            angle.Value = System.Math.Atan(imaginary_number / real_number);
            angle.AngleType = AngleType_e.Radians;

            double angle_rad = angle.ConvertTo(AngleType_e.Radians);

            if (real_number > 0 && imaginary_number > 0)
            {
                //ربع 1
                //الزاوية نفسها بدون تغيير//
            }

            else if (real_number < 0 && imaginary_number > 0)
            {
                //ربع 2
                angle.Value = Constants.PI - System.Math.Abs(angle_rad);
            }

            else if (real_number < 0 && imaginary_number < 0)
            {
                //ربع 3
                angle.Value = Constants.PI + angle_rad;
            }

            else
            {
                //ربع 4
                angle.Value = 2 * Constants.PI - System.Math.Abs(angle_rad);
            }

            return new PolarComplexNumber(modulus, angle);
        }
```

```
        ///
        public override void RefreshNumber()
        {
            if (real_number != null)
                real_number = System.Math.Round(real_number, this.Precision);
            if (imaginary_number != null)
                imaginary_number =
                    System.Math.Round(imaginary_number, this.Precision);
        }

        ///
        public override string ToString()
        {
            string s = "0";
            this.RefreshNumber();

            if (real_number != 0 && imaginary_number != 0)
            {
                if (imaginary_number > 0)
                    s = string.Format("{0} + i {1}",real_number,imaginary_number);
                else
                    s =string.Format("{0} - i {1}",real_number,-imaginary_number);
            }

            else if (real_number == 0 && imaginary_number != 0)
            {
                if (imaginary_number > 0)
                    s = string.Format("i {0}", imaginary_number);
                else
                    s = string.Format("- i {0}", -imaginary_number);
            }

            else if (real_number != 0 && imaginary_number == 0)
                s = string.Format("{0}", real_number);

            return s;
        }

        static RectangularComplexNumber Addition
            (RectangularComplexNumber num1, RectangularComplexNumber num2)
        {
            //a1+b1i + a2+b2i = (a1+a2)+(b1+b2)i
            return
                new RectangularComplexNumber(num1.RealNumber + num2.RealNumber,
                    num1.ImaginaryNumber + num2.ImaginaryNumber);
        }

        static RectangularComplexNumber Subtraction
            (RectangularComplexNumber num1, RectangularComplexNumber num2)
        {
            //a1+b1i - a2+b2i = (a1-a2)+(b1-2)i
            return
                new RectangularComplexNumber(num1.RealNumber - num2.RealNumber,
                    num1.ImaginaryNumber - num2.ImaginaryNumber);
        }
```

```
static RectangularComplexNumber Multiplication
    (RectangularComplexNumber num1, RectangularComplexNumber num2)
{
    //(a1+b1i)*(a2+b2i)=a1a2 + a1b2i + a2b1i - b1b2=
    //(a1a2+a1b2i) + (-b1b2 + a2b1i) = (a1a2-b1b2)+(a1b2 + a2b1)i
    return
        new RectangularComplexNumber(
            num1.RealNumber *
            num2.RealNumber -
            num1.ImaginaryNumber *
            num2.ImaginaryNumber,
            num1.RealNumber *
            num2.ImaginaryNumber +
            num2.RealNumber *
            num1.ImaginaryNumber);
}

static RectangularComplexNumber Division
    (RectangularComplexNumber num1, RectangularComplexNumber num2)
{
    //(a1+b1i)/(a2+b2i) = (a1+b1i)(a2-b2i)/(a2^2 + b2^2) =
    // (a1a2+a1b2i+a2b1i-b1b2)/(a2^2 + b2^2)=
    //((a1a2-b1b2)/(a2^2 + b2^2))+((a1b2+a2b1)/(a2^2 + b2^2))i
    double a1 = num1.RealNumber, a2 = num2.RealNumber, b1 =
        num1.ImaginaryNumber, b2 = num2.ImaginaryNumber;
    return
        new RectangularComplexNumber(
            (a1 * a2 + b2 * b1) / (a2 * a2 + b2 * b2),
            (a2 * b1 - a1 * b2) / (a2 * a2 + b2 * b2));
}
#endregion


#region Operators
public static RectangularComplexNumber
    operator +(RectangularComplexNumber n1, RectangularComplexNumber n2)
{
    return Addition(n1, n2);
}

public static RectangularComplexNumber
    operator +(RectangularComplexNumber n1, ComplexNumber n2)
{
    RectangularComplexNumber num1, num2;
    num1 = (RectangularComplexNumber)n1;

    if (n2 is RectangularComplexNumber)
        num2 = (RectangularComplexNumber)n2;
    else
        num2 = ((PolarComplexNumber)n2).ToRectangular();

    return Addition(num1, num2);
}

public static RectangularComplexNumber
    operator +(ComplexNumber n1, RectangularComplexNumber n2)
{
    RectangularComplexNumber num1, num2;
    num2 = (RectangularComplexNumber)n2;
```

```
            if (n1 is RectangularComplexNumber)
                num1 = (RectangularComplexNumber)n1;
            else
                num1 = ((PolarComplexNumber)n1).ToRectangular();

            return Addition(num1, num2);
        }

        public static RectangularComplexNumber
            operator -(RectangularComplexNumber n1, RectangularComplexNumber n2)
        {
            return Subtraction(n1, n1);
        }

        public static RectangularComplexNumber
            operator -(RectangularComplexNumber n1, ComplexNumber n2)
        {
            RectangularComplexNumber num1, num2;
            num1 = (RectangularComplexNumber)n1;

            if (n2 is RectangularComplexNumber)
                num2 = (RectangularComplexNumber)n2;
            else
                num2 = ((PolarComplexNumber)n2).ToRectangular();

            return Subtraction(num1, num2);
        }

        public static RectangularComplexNumber
            operator -(ComplexNumber n1, RectangularComplexNumber n2)
        {
            RectangularComplexNumber num1, num2;
            num2 = (RectangularComplexNumber)n2;

            if (n1 is RectangularComplexNumber)
                num1 = (RectangularComplexNumber)n1;
            else
                num1 = ((PolarComplexNumber)n1).ToRectangular();

            return Subtraction(num1, num2);
        }

        public static RectangularComplexNumber
            operator *(RectangularComplexNumber n1, RectangularComplexNumber n2)
        {
            return Multiplication(n1, n2);
        }

        public static RectangularComplexNumber
            operator *(RectangularComplexNumber n1, ComplexNumber n2) {
            RectangularComplexNumber num1, num2;
            num1 = (RectangularComplexNumber)n1;

            if (n2 is RectangularComplexNumber)
                num2 = (RectangularComplexNumber)n2;
            else
                num2 = ((PolarComplexNumber)n2).ToRectangular();

            return Multiplication(num1, num2);
        }
```

```
        public static RectangularComplexNumber
            operator *(ComplexNumber n1, RectangularComplexNumber n2)
        {
            RectangularComplexNumber num1, num2;
            num2 = (RectangularComplexNumber)n2;

            if (n1 is RectangularComplexNumber)
                num1 = (RectangularComplexNumber)n1;
            else
                num1 = ((PolarComplexNumber)n1).ToRectangular();

            return Multiplication(num1, num2);
        }

        public static RectangularComplexNumber
            operator /(RectangularComplexNumber n1, RectangularComplexNumber n2)
        {
            return Division(n1, n2);
        }

        public static RectangularComplexNumber
            operator /(RectangularComplexNumber n1, ComplexNumber n2)
        {
            RectangularComplexNumber num1, num2;
            num1 = (RectangularComplexNumber)n1;

            if (n2 is RectangularComplexNumber)
                num2 = (RectangularComplexNumber)n2;
            else
                num2 = ((PolarComplexNumber)n2).ToRectangular();

            return Division(num1, num2);
        }

        public static RectangularComplexNumber
            operator /(ComplexNumber n1, RectangularComplexNumber n2)
        {
            RectangularComplexNumber num1, num2;
            num2 = (RectangularComplexNumber)n2;

            if (n1 is RectangularComplexNumber)
                num1 = (RectangularComplexNumber)n1;
            else
                num1 = ((PolarComplexNumber)n1).ToRectangular();

            return Division(num1, num2);
        }

        public static RectangularComplexNumber
            operator ^(RectangularComplexNumber n1, int n2)
        {
            RectangularComplexNumber res = n1;

            if (n2 == 0)
                res = new RectangularComplexNumber(1, 0);
            else if (n2 > 0)
                for (int i = 1; i < n2; i++)
                    res *= n1;
            else
                for (int i = -1; i > n2; i--)
```

```
                    res /= n1;

                return res;
            }
            #endregion
        }
}
```

العدد العقدي القطبي (المثلثي)

إذا كان العدد العقدي قطبيًّا، فإنه عبارة عن شعاع، له طويلة، ويميل بزاوية (تسمى الطور Phase).



```
namespace Eng27.CodeBank.Math
{
    ///
    public class PolarComplexNumber : ComplexNumber
    {
        #region Constructors
        ///
        public PolarComplexNumber()
        {

        }

        ///
        public PolarComplexNumber(double Magnitude, Angle Angle)
        {
            magnitude = Magnitude;
            phase = Angle;
            this.RefreshNumber();
        }

        ///
        public PolarComplexNumber(double Magnitude, double AngleInDegrees)
        {
            magnitude = Magnitude;
            phase = new Angle(AngleInDegrees, AngleType_e.Degrees);
            this.RefreshNumber();
        }
        #endregion

        #region Properties
        private double magnitude;
        ///
        public double Magnitude
        {
            get { return magnitude; }
            set
            {
                magnitude = value;
                this.RefreshNumber();
            }
        }
```

```
        private Angle phase;
        ///
        public Angle Phase
        {
            get { return phase; }
            set
            {
                phase = value;
                this.AngleType = phase.AngleType;
                this.RefreshNumber();
            }
        }
        #endregion


        #region Methods
        ///
        public RectangularComplexNumber ToRectangular()
        {
            double real, imaginary;
            real = magnitude *
                System.Math.Cos(phase.ConvertTo(AngleType_e.Radians));
            imaginary = magnitude *
                System.Math.Sin(phase.ConvertTo(AngleType_e.Radians));
            return new RectangularComplexNumber(real, imaginary);
        }

        ///
        public override void RefreshNumber() {
            if (magnitude != null)
                magnitude = System.Math.Round(magnitude, this.Precision);
            if (phase != null)
                phase.Value = System.Math.Round(phase.Value, this.Precision);
        }

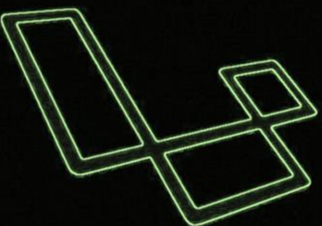
        ///
        public override string ToString()
        {
            string s = "0";
            this.RefreshNumber();
            double a =
               System.Math.Round(phase.ConvertTo(this.AngleType), this.Precision);

            if (this.AngleType == AngleType_e.Degrees)
                s = string.Format("[{0}, {1}\u00b0]", magnitude, a);
            else
                s = string.Format("[{0}, {1}]", magnitude, a);
            return s;
        }
        #endregion


        #region Operators
        public static PolarComplexNumber
            operator *(PolarComplexNumber num1, PolarComplexNumber num2)
        {
            //[r1 * r2, a1 + a2]
            return new PolarComplexNumber
                (num1.Magnitude * num2.Magnitude, num1.Phase + num2.Phase);
        }
```

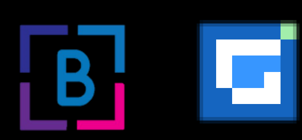
```
        public static PolarComplexNumber
            operator /(PolarComplexNumber num1, PolarComplexNumber num2)
        {
            //[r1 / r2, a1 - a2]
            return new PolarComplexNumber
                (num1.Magnitude / num2.Magnitude, num1.Phase - num2.Phase);
        }

        public static PolarComplexNumber
            operator ^(PolarComplexNumber n1, int n2)
        {
            return new PolarComplexNumber
                (System.Math.Pow(n1.Magnitude, n2), n1.Phase.Value * n2);
        }
        #endregion
    }
}
```



الباب الرابع

فريموركات جاهزة





أعتذر كبداية من القرّاء – النحويّين منهم خصوصًا – على تسمية هذا الباب بهذا الشكل، ولكن للضرورة أحكام كما يقال.

تكلمنا في الفصول السابقة عن ماهية الأدوات في البرمجة – بشكل معمّق – وكيفية تصميمها وإنشاءها، والآن سنتناول أدوات جاهزة ستضفي على برامجك لمسات سحرية، يمكن لهذه الأدوات أن توفر عليك جهدًا كبيرًا في تصميم برامجك، وقد تم إنشاؤها تمامًا كما في الفصول السابق.

هناك الكثير من منصات أو أُطُر العمل – أو الـ Frameworks كما عُنوِنَ به هذا الفصل – والتي تقدم لك أدوات كثيرة، منها ما هو مجانيّ ومنها المدفوع. سأتناول في هذا الباب كلًّا من DevExpress وMetro وBunifu وXander وGuna.UI ومكتبة Transitions، آملًا أن تحصل على ما توقعتَه من هذا الكتاب عند انتهاءك من فصول هذا الباب.

الفصل التاسع – مدخل إلى تصميم النوافذ

يعطيك الفيجوال ستوديو إمكانيات كبيرة في تصميم النوافذ، فعلى عكس تصميم الأدوات والتي غالبًا ما كانت تحاك فيها مجريات الأمور في الكود، فالنوافذ تصمم بشكل تفاعلي، بالإضافة للكود، وهذا ما ناقشناه في الفصل الثاني في بداية الكتاب، في الفقرة **مصمم النموذج designer**.

فصول هذا الباب موجه نحو تصميم النوافذ، باستخدام منصات جاهزة بالتحديد. كانت عندي نية في إنشاء فصول تتحدث عن تصميم النوافذ باستخدام منصة دوت نت(بالاعتماد على الأدوات القياسية) إلا أني رأيت أن التصاميم الموجودة في الكتاب الأول كافية، ولو أنها محدود. كما أن هذه الفصول التي كنت أنوي إنشاءها كان من

المفترض أن تحوي بعض التصاميم الشائعة التقليدية، أو بعض التصاميم بالاعتماد على مكتبة Eng27 التي تناولناها بالتفصيل في الفصل السابع.

من المفيد أن نستعرض في البداية الاعتبارات التصميمية التي يجب عليك أخذها بعين الاعتبار قبل البدء بفصول هذا الباب، إلا أنني لن أفعل ذلك، وسأنتقل مباشرة لتصميم النوافذ، من خلال المكتبات والمنصات الجاهزة، وذلك لسببين:

- أنني لن أناقش تصميم النوافذ من خلال الأدوات التقليدية، بل على العكس، سأفترض وجود خبرة عندك فيها.
- وجود دورة [1] جيدة تفي بالغرض، من قناة تكنو يو Techo U.

ولكنني بالمقابل سأبدأ بمكتبة يمكنك الاعتماد عليها عند تصميم نوافذ تطبيقاتك لإضفاء الحركة على أدواتها، كمدخل لتصميم النوافذ.

عمومًا، عند تصميم النوافذ فإن جُل ما تقوم به هو العمل مع:

- صندوق الأدوات Toolbox.
- نافذة الخصائص Property Window.
- متصفح المشروع Solution Explorer.
- نافذة العمل، والتي فيها إما الكود أو النافذة أو أمور أخرى.

من الأمور التي كنت أنوي أن أغطيها في هذا الكتاب هو أهم الخصائص والأحداث والطرق للأدوات الأكثر شيوعًا، أي محتوى نافذة الخصائص، بشكل مشابه لفقرة الأدوات الأكثر شيوعًا في الفصل الأول، ولكنني تراجعت عنها، لافتراضي وجود خبرة لا بأس بها عند القارئ تمكنه من التفريق بين الخصائص والأحداث المختلفة ومعرفة الخاصية والحدث المناسبين للأماكن المناسبة.

[1] دورة مهارات تصميم البرامج لقناة تكنو يو
https://www.youtube.com/playlist?list=PLhiFu-f80eo9XgONQ1NcLoi_8MTo3ziOm

مكتبة Transitions

هي مكتبة صغيرة تُضفي على برامجك حيوية جميلة، تعطي أدواتك إمكانية التنقل والتحرك من مكان لآخر، بمعنى آخر: تُحرِّك ساكنًا، مما يجعل تصميم UI في تطبيقاتك أفضل. تتميز المكتبة ببساطتها وصغر حجمها، فصحيح أن فئاتها قليلة، إلا أن إمكانياتها كبيرة بفضل أنواع البيانات العديدة التي تدعمها. يمكنك تحميلها بالبحث في غوغل عن dotnet transitions، كما يمكنك ذلك من خلال NuGet.[1]

للمكتبة فئة واحدة رئيسية، وفئات أخرى تمثل أنواع النقلات. الفئة الرئيسية لها طريقتين، ولكائنات الفئة طريقتين[2]. يتم تنفيذ النقلة Transition خلال فترة زمنية مقدرة بالـ ms، وبعد تزويد الفئة – أو كائناتها – بالأداة المطلوب إجراء النقلة عليها واسم الخاصية المراد التأثير عليها، وقيمة الخاصية بعد التنفيذ، وإذا كنت تطبق الأكواد على الفئات نفسها وليس على كائناتها فيمكنك إضافة القيمة قبل التنفيذ.

وأنواع البيانات التي يمكن للمكتبة التأثير عليها هي Int وFloat وDouble وColor و String، هذا يعني أن الخصائص التي يمكنك إجراء النقلات عليها هي على سبيل المثال Text وWidth وLeft وForeColor وما شابه هذه الخصائص.

أنواع النقلات

عند إنشاء نقلة يجب عليك تحديد نوعها، وهذا ما يحدد شكل النقلة وبارامترات أخرى مطلوبة لإجراء النقلة. النقلات الافتراضية هي:

- نقلة خطية TransitionType_Linear:
  هي نقلة تتغير بشكل ثابت.

---

[1] راجع الروابط:
من موقع Github:
https://github.com/UweKeim/dot-net-transitions/blob/master/Bin/Transitions/Transitions.dll?raw=true
من موقع Nugut:
https://www.nuget.org/packages/dot-net-transitions/

[2] أقصد بالفئة عند استخدامها مباشرة دون استنساخ كحالة الكود Console.ReadKey، وبكائن الفئة عند استخدام الفئة بعد الاستنساخ (بعد إنشاء كائن ما، يتم التعامل معه للوصول لمكونات الفئة).

- نقلة متسارعة TransitionType_Acceleration:

تبدأ بسرعة معدومة، وتتزايد لتصل للسرعة القصوى عند نهاية النقلة.

- نقلة متناقصة TransitionType_Deceleration:

تبدأ بالسرعة القصوى، وتتناقص حتى تصل للصفر عند نهاية النقلة.

- نقلة مُخمّدة بشكل حاد TransitionType_CriticalDamping:

هي نقلة تنتهي بشكل مفاجئ.

- نقلة متزايدة متناقصة TransitionType_EaseInEaseOut:

تبدأ بسرعة معدومة، تتزايد حتى تصل للسرعة القصوى في منتصف زمن النقلة، ثم تعود لتتناقص حتى تصل للصفر في نهاية زمن النقلة.

- نقلة مرتدة 🏐 TransitionType_Bounce:

تتسارع لتصل للقيمة المطلوبة خلال نصف زمن النقلة ثم تعود للقيمة الأصل مجددًا. أشبه بكرة تتسارع باتجاه الجاذبية، ثم ترتد بعد اصطدامها بالأرض بعكس الجاذبية.

- نقلة غير مرتدة 🤷‍♂️ TransitionType_ThrowAndCatch:

بعكس النقلة المرتدة، تتباطأ لتصل للقيمة المطلوبة ثم تعود متسارعة للقيمة الأصلية.

- نقلة الومضة TransitionType_Flash:

تعطيك إمكانية تكرار النقلة على شكل ومضات، لذلك فعليك تحديد عدد الومضات ومدة كل ومضة.

- نقلة خاصة TransitionType_UserDefined:

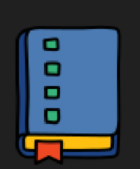
هي نقلة مكونة من أكثر من نقلة.

تهيئة المشروع

بعد تحميل نسخة من المكتبة، أضفها لمراجع مشروعك، ثم صرّح عنها بالكلمة using.

إنشاء النقلات

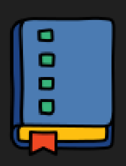
إذا أردت إنشاء نقلة لخاصية واحدة فيمكنك استخدام فئة Transition مباشرةً دون الحاجة لإنشاء نسخة منها، وذلك كما يلي:



```
Transition.run(target, strPropertyName, destinationValue, transitionMethod);
```

حيث: target الأداة المراد إجراء النقلة عليها، وstrPropertyName قيمة نصية تمثل اسم الخاصية المراد التأثير عليها، وdestinationValue القيمة المراد الوصول إليها، وtransitionMethod طريقة تنفيذ النقلة (نوع النقلة).

كما يمكنك تحديد القيمة الابتدائية كما يلي:



```
Transition.run(target, strPropertyName, initialValue, destinationValue,
    transitionMethod);
```

أما إذا أردت التأثير على أكثر من خاصية بالوقت نفسه، فاستخدم الصيغة التالية:



```
Transition t = new Transition(transitionMethod);
t.add(target1, strPropertyName1, destinationValue1);
t.add(target2, strPropertyName2, destinationValue2);
.
.
t.run();
```

حدث اكتمال النقلة TransitionCompletedEvent

بعد تنفيذ النقلة بالكامل – وذلك بانقضاء زمنها – يُفجَّر [1] هذا الحدث في حال أضفته لأكوادك، ومن خلاله يمكنك تنفيذ نقلات متلاحقة أو القيام ببعض الأمور بعد تنفيذ النقلات.

[1] كلمة تفجير عند استخدامها مع الأحداث يكون معناها حدوث الحدث، وهي ترجمة حرفية للمصطلح البرمجي المعبر عن حدوث أو حصول الحدث.

فلَو كان لديك كائنًا من نوع Transition باسم a مثلًا، فإن إنشاء الحدث يكون كالتالي:



```
a.TransitionCompletedEvent += a_TransitionCompletedEvent;
.
.
void a_TrnsitionCompletedEvent(object sender, Trnsition.Args e)
{
    //الأكواد التي ترغب بتنفيذها بعد تنفيذ النقلة بالكامل
}
```

أمثلة

في البداية أنصحك بتحميل المثال الرسمي للمكتبة من صفحتها على GitHub والاطلاع عليه وفهمه. ثم تابع الأمثلة التالية:

- لتغيير لون النافذة عند النقر على زر:



```
private void button1_Click(object sender, EventArgs e)
{
    Transition.run(this, "BackColor", Color.Red, new TransitionType_Linear(1000));
}
```

الكلمة this تمثل النافذة الحالية (في الواقع هذه الكلمة تمثل الفئة المحتواة فيها، وعلى اعتبار النوافذ فئات فهي تمثل النافذة).

- لتغيير حجم صندوق النصوص عند إعطائه التركيز أو سحبه منه:



```
private void textBox1_Enter(object sender, EventArgs e)
{
    Transition.run(sender, "Width", 200, new TransitionType_Acceleration(1000));
}

private void textBox1_Leave(object sender, EventArgs e)
{
    Transition.run(sender, "Width", 100, new TransitionType_Acceleration(700));
}

private void textBox2_Enter(object sender, EventArgs e)
{
    Transition.run(sender, "Width", 200, new TransitionType_Acceleration(1000));
}

private void textBox2_Leave(object sender, EventArgs e)
{
    Transition.run(sender, "Width", 100, new TransitionType_Acceleration(700));
}
```

اعتمدنا على أن sender يحمل في جعبته بيانات تمثل خصائصه، وماهيته، ولولا ذلك لأرسلنا textBox1 وtextBox2 كوسطاء للإجراء run.

- لنقل أداة من موقع لآخر:



```
Transition a = new Transition(new TransitionType_Deceleration(1000));
a.add(button1, "Left", 200);
a.add(button1, "Top", 300);
a.run();
```

- لتنفيذ نقلتين بعد بعضهما، بالاعتماد على حدث اكتمال النقلة:



```
private void Form1_Load(object sender, EventArgs e)
{
    Transition a = new Transition(new TransitionType_Deceleration(1000));
    a.add(button1, "Width", 200);
    a.add(button1, "Height", 100);
    a.run();
    a.TransitionCompletedEvent += a_TransitionCompletedEvent;
}

private void a_TransitionCompletedEvent(object sender, Transition.Args e)
{
    Transition.run(button1,
        "BackColor",
        Color.Red,
        new TransitionType_Flash(5, 300));
}
```

- لإنشاء نقلة مخصصة UserDefined:



```
var elements = new List<TransitionElement>();
elements.Add(new TransitionElement(50, 50, InterpolationMethod.Accleration));
elements.Add(new TransitionElement(75,100, InterpolationMethod.Linear));

Transition.run(button1,
    "BackColor",
    Color.Green,
    new TransitionType_UserDefined(elements, 2000));
```

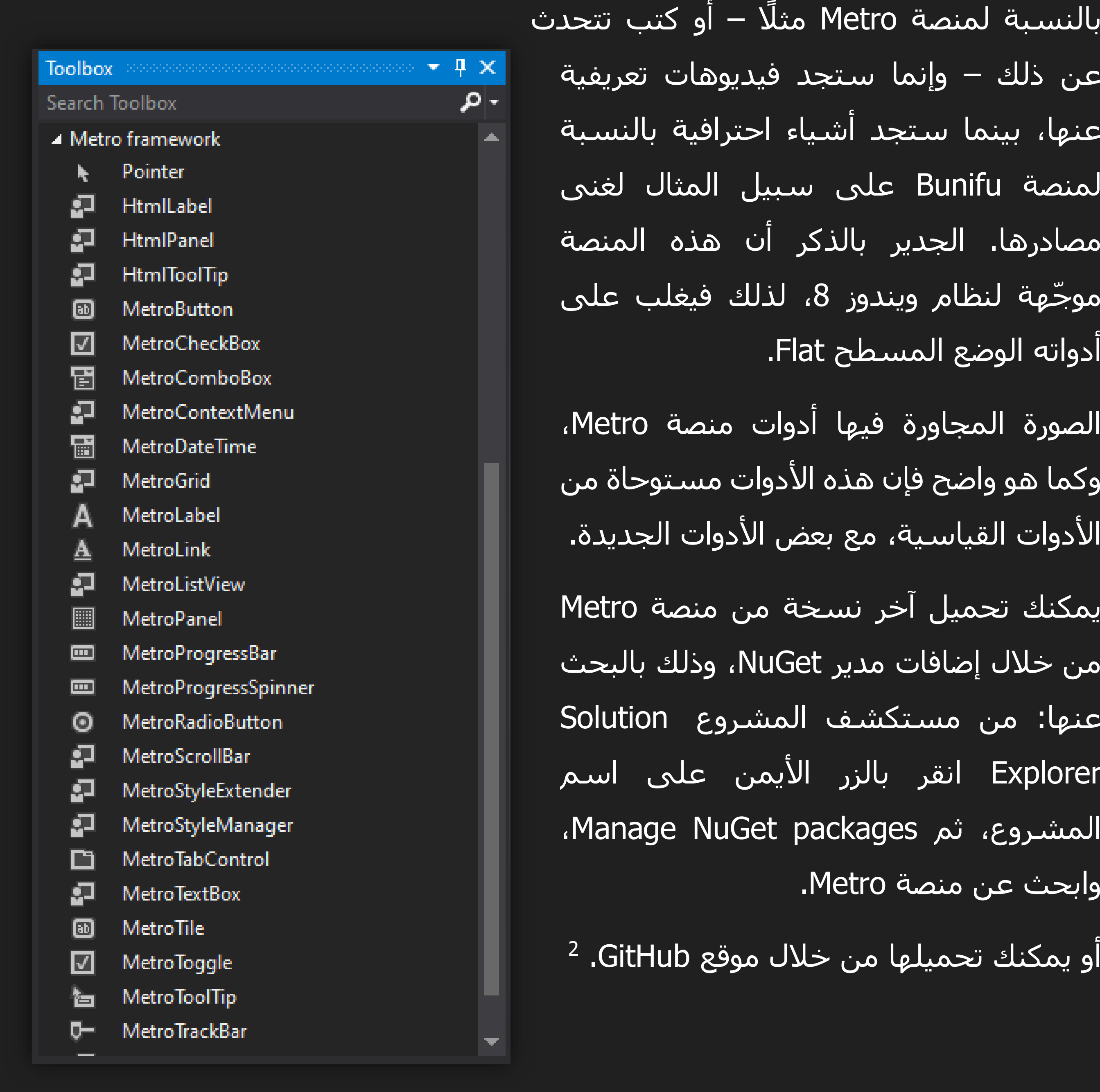
الفصل العاشر – منصات صغيرة

كتبت هذا الفصل – والفصل التالي – قبل فترة طويلة من إنهائي للكتاب (قبل عام تقريبًا)، لهذا فإنك قد تجد بعض الأدوات – أو المنصات عمومًا – قديمة وقد ظهرت تحديثات لها، خصوصًا إذا كان لديك اطلاع جيد على الموضوع. كما أن هناك بعض المنصات التي كنت أنوي تناولها في هذا الكتاب ولم أوفّق لذلك.

منصة Metro

من أشهر أطر العمل، مجانية، غنية بالأدوات، عصرية، بأدوات أنيقة، متعددة الاستخدامات، بسيطة التصميم، سهلة الاستخدام.

من مساوئها عدم وجود مصادر أو مراجع Documentations خاصة بالشركة المبرمجة يُرجَع إليها عند عدم معرفة كيف تُستخدم الأدوات[1]، وبالمقابل أعلم أنك كمبرمج حصّل علمه من اليوتيوب – في حال كنت من هذه النوعية من المبرمجين – لا تأبه للمصادر ولا تهتم لها ومع ذلك فهي بطريقة غير مباشرة تهمك، لأن غياب مصادر الجهة المبرمجة يجعل من الفيديوهات والكتب التي تتناول الموضوع أقل. فلن تجد فيديوهات احترافية بالنسبة لمنصة Metro مثلاً – أو كتب تتحدث عن ذلك – وإنما ستجد فيديوهات تعريفية عنها، بينما ستجد أشياء احترافية بالنسبة لمنصة Bunifu على سبيل المثال لغنى مصادرها. الجدير بالذكر أن هذه المنصة موجّهة لنظام ويندوز 8، لذلك فيغلب على أدواته الوضع المسطح Flat.



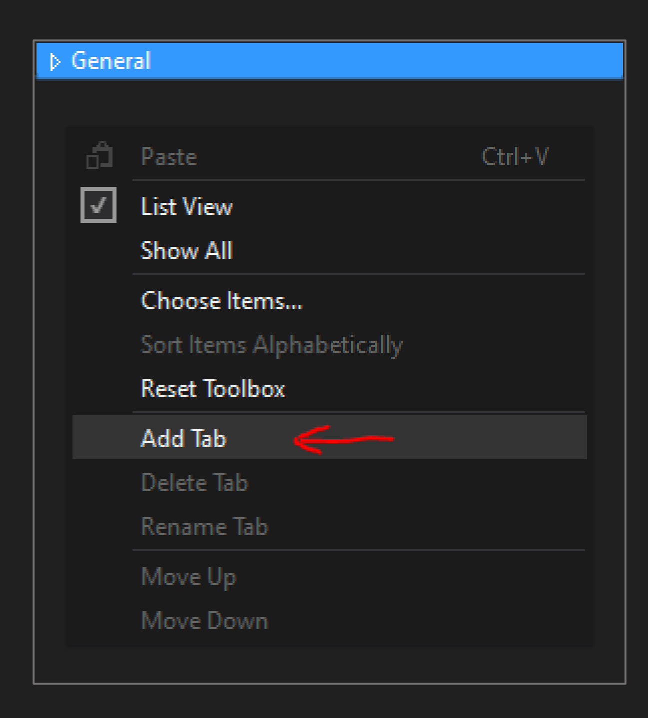
الصورة المجاورة فيها أدوات منصة Metro، وكما هو واضح فإن هذه الأدوات مستوحاة من الأدوات القياسية، مع بعض الأدوات الجديدة.

يمكنك تحميل آخر نسخة من منصة Metro من خلال إضافات مدير NuGet، وذلك بالبحث عنها: من مستكشف المشروع Solution Explorer انقر بالزر الأيمن على اسم المشروع، ثم Manage NuGet packages، وابحث عن منصة Metro.

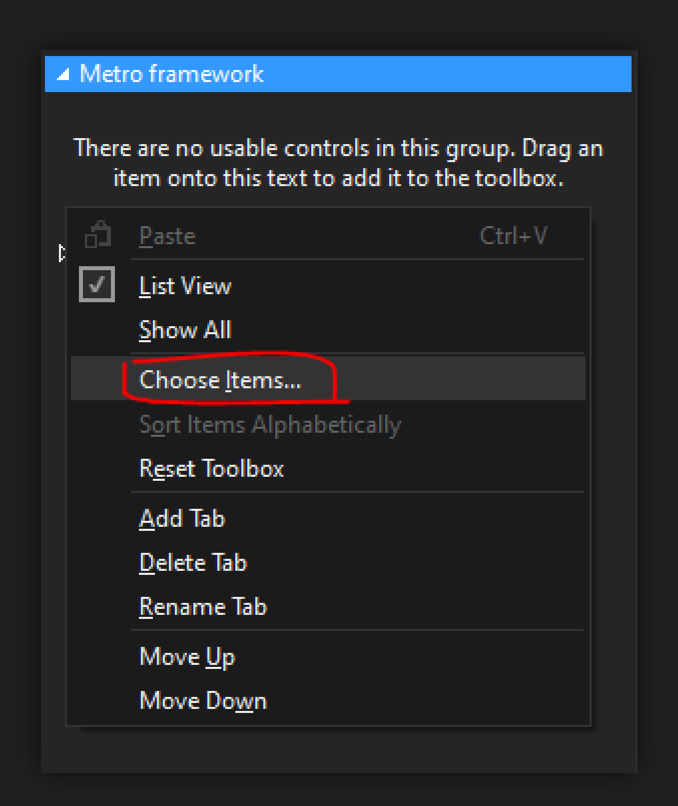
أو يمكنك تحميلها من خلال موقع GitHub.[2]

[1] لكن عمومًا، راجع هذا الموقع http://denricdenise.info/.
[2] يمكنك تحميلها من الرابط https://thielj.github.io/MetroFramework/.

بعد تحميل ملفات المنصة، قم بإدراج تبويب جديد ضمن صندوق الأدوات ToolBox:



سمّها Metro framework مثلًا ثم أضف الأدوات:



انقر على تصفح Browse ثم أوجد المنصة (وهي ملف من نوع dll كما تعلم):

Choose Toolbox Items

Silverlight Components | Windows XAML Components | WPF Components
.NET Framework Components | COM Components | System.Activities Components

	Name	Namespace	Assembly Name
☑	AccessDataSource	System.Web.UI.WebControls	System.Web
☐	Activity	System.Workflow.ComponentModel	System.Workflow.ComponentMo
☐	AddGroupControl	Microsoft.TeamFoundation.Controls.WinForms	Microsoft.TeamFoundation.Contr
☐	ADODC	Microsoft.VisualBasic.Compatibility.VB6	Microsoft.VisualBasic.Compatibili
☐	ADODCArray	Microsoft.VisualBasic.Compatibility.VB6	Microsoft.VisualBasic.Compatibili
☐	AdRotator	System.Web.UI.MobileControls	System.Web.Mobile
☑	AdRotator	System.Web.UI.WebControls	System.Web
☑	AppearanceEditorPart	System.Web.UI.WebControls.WebParts	System.Web
☐	AssemblyInstaller	System.Configuration.Install	System.Configuration.Install

Filter: Clear

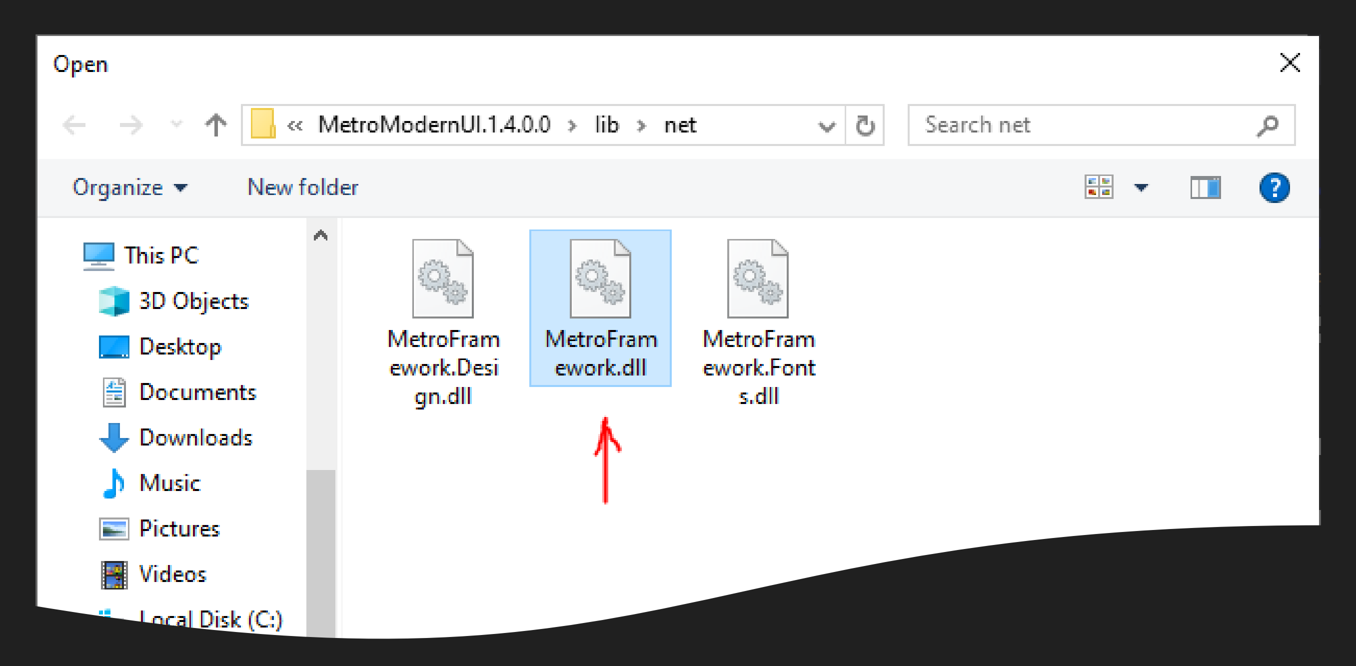
AccessDataSource Browse...

Language: Invariant Language (Invariant Country)

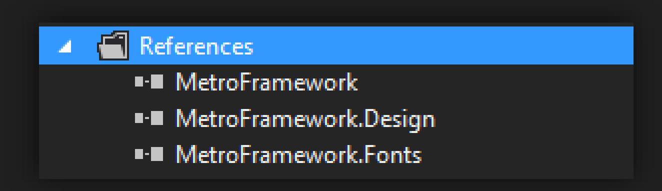
Version: 4.0.0.0

OK Cancel Reset

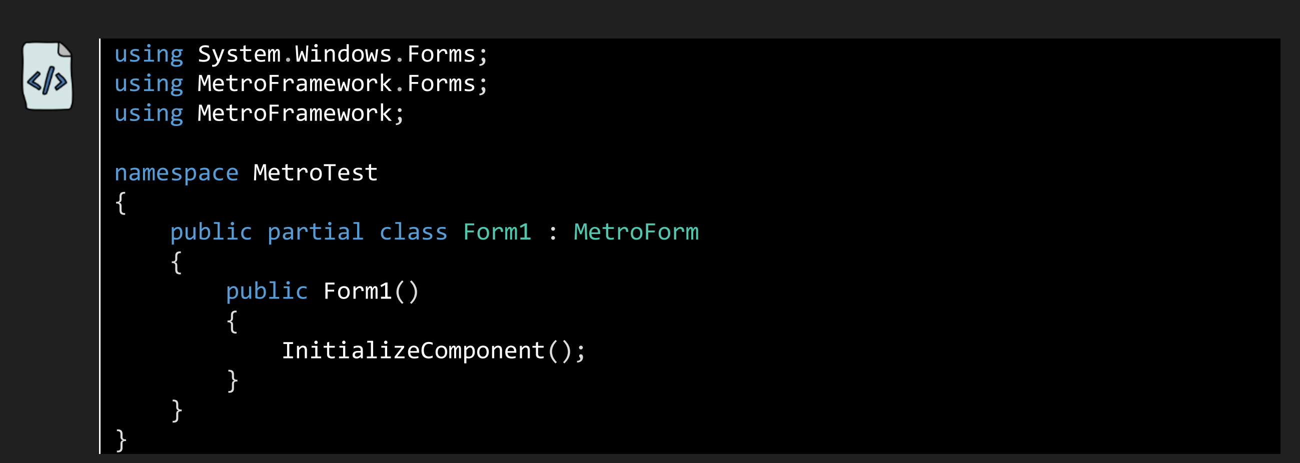
اختر الملف الرئيسي:



قد تواجه مشكلة أثناء تحميل الأدوات (الخطوة الأخيرة)، غيّر مكان المكتبة ضمن القرص الصلب، إذا تكررت المشكلة جرّب مكانًا مختلفًا ضمن القرص الصلب (كرّر هذه الخطوة حتى يتم التحميل بنجاح). ثم بعد إدراج الأدوات لصندوق الأدوات، أسند المكتبة لمراجع المشروع References كما في الشكل المجاور.



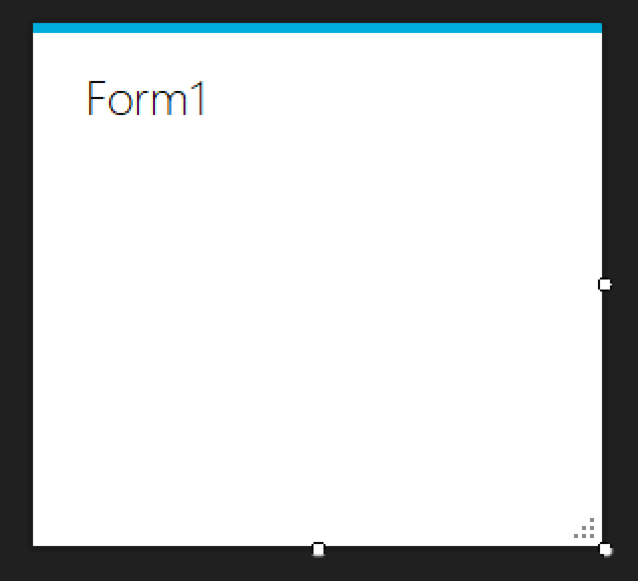
وكخطوة أخيرة، عدّل كود البرنامج ليصبح بالشكل التالي:



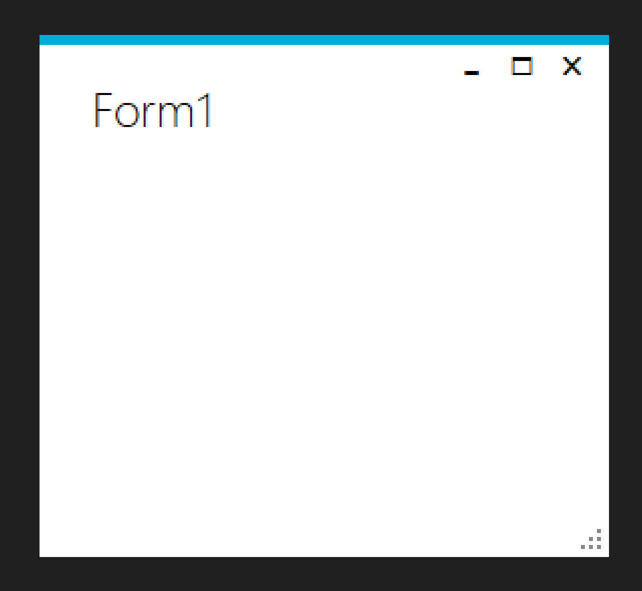
```
using System.Windows.Forms;
using MetroFramework.Forms;
using MetroFramework;

namespace MetroTest
{
    public partial class Form1 : MetroForm
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

أصبحت نافذة البرنامج في طور التصميم بالشكل:



وعند تنفيذ البرنامج:



ما يميز نوافذ Metro – كما هو واضح من الصورتين السابقتين – أن تغيير حجم النافذة يكون من زاوية واحدة، وأن هناك خطًا بلون أزرق خفيف يزيّن النافذة من أعلاها، وأن عنوان النافذة – الذي يُضبط من خلال الخاصية Text – موضوع بطريقة مختلفة عن المعتاد (عادة ما يكون النص Text ضمن شريط في أعلى النافذة).

لن نقوم بشرح أدوات هذه المنصة – والمنصات اللاحقة – وإنما سنقتصر على ذكر بعض الأمثلة – والتي يمكن أن ترقى لكونها مشاريع جاهزة بسيطة – لكيفية إنشاء برمجيات صغيرة مفيدة بتصاميم أنيقة يمكن الاستفادة منها في برامج أكبر وأشمل، في حين أنه قد نشرح بعض الأدوات الجديدة غير الموجودة بين الأدوات القياسية التي ذكرناها في الفصل الثاني.

بداية سريعة – صناديق الرسائل، وبعض الإعدادات

صمم النافذة التالية:

الأدوات المستخدمة هي أربعة أدوات عناوين metroLabel، وصندوق نص metroTextBox، وقائمتين منسدلتين metroComboBox، ومفتاح metroToggle، وستة لوائح metroTile، وأداة تلميح metroToolTip، ومتصيد أخطاء errorProvider، والنموذج MetroForm بطبيعة الحال.

اضبط خصائص الأدوات كما يلي:

<table>
<tr><th>الأداة</th><th>الخاصية</th><th>القيمة</th></tr>
<tr><td rowspan="5">Form1</td><td>AutoScaleMode</td><td>None</td></tr>
<tr><td>MaximizeBox</td><td>False</td></tr>
<tr><td>MinimizeBox</td><td>False</td></tr>
<tr><td>Resizable</td><td>False</td></tr>
<tr><td>ShadowType</td><td>DropShadow</td></tr>
</table>

	Size	382 ,677
	Text	Metro MessageBox by Eng27
	TextAlign	Center
metroComboBox1	Size	29 ,241
metroComboBox2	Size	29 ,241
metroTextBox1	Size	23 ,241
	WaterMark	Type your name here
metroTile1	Text	AbortRetryIgnore
metroTile2	Text	OK
metroTile3	Text	OKCancel
metroTile4	Text	RetryCancel
metroTile5	Text	YesNo
metroTile6	Text	YesNoCancel
metroToggle1	Checked	True

على اعتبار أن هذا الفصل – وهذا الكتاب عمومًا – موجّه لمناقشة أساليب التصميم أكثر من أساليب التكويد [1]، فسنناقش الأدوات وجديدها عند كل أداة نستخدمها، كما سننوه للجديد والغريب من الأكواد كتعليق على الكود. لذلك فسنناقش الخصائص المميزة للأدوات المستخدمة والتي لا يعرفها من اعتاد على الأدوات القياسية. ومن المحتمل أن هذا الذي اعتاد على الأدوات القياسية أنه كان يبحث عن هذه الخصائص أو أنه بذل جهدًا في تأليف ودمج الأكواد من هنا وهناك للحصول على نتيجة معينة.

وخير ما نبدأ به هو نافذة البرنامج، صحيح أنها باسم Form1 لكنها من النوع MetroForm، أي أن البعض قد يستغرب عدم كون اسمها metroForm1، والسبب يا صديقي أننا أنشأنا

[1] التكويد: إنشاء الأكواد.

المشروع بدايةً باستخدام الأدوات القياسية ثم غيرنا مصدر الوراثة من Form إلى MetroForm دون تغيير اسم النافذة ككائن.

واللافت ضمن خصائص نوافذ هذه المنصة هو وجود خصائص مثل Resizable وShadowType وTextAlign، والتي تختصر كثيرًا عليك، فالأولى كنا نحصل عليها بجعل كل من الخصائص MaximumSize وMinimumSize وSize متساوية، والثانية لا وجودَ لها أصلًا ضمن الخصائص القياسية، والثالثة أيضًا. كما يوجد خصائص جديدة مثل Theme وStyle وMovable وغيرها، وكشرح مختصر للخصائص الستة السابقة:

- الخاصية Resizable تعطي نافذة برنامجك إمكانية تغيير الحجم.
- الخاصية ShadowType تعطي نافذة برنامجك ظلًّا يختلف عن الظل الافتراضي.
- الخاصية TextAlign تضبط مكان عنوان النافذة.
- الخاصية Theme تضبط سِمة النافذة، وسنفصّلها لاحقًا إن شاء الله تعالى.
- الخاصية Style تضبط الألوان الثانوية في التطبيق (كاللون الأزرق أعلى النافذة).
- الخاصية Movable تعطي نافذة برنامجك إمكانية النقل، وهي ما كنا نبذل جهدًا للوصول إليها عند التعامل مع النوافذ كـ Panel (النوافذ عديمة الحواف)

أما صناديق اللوائح المسندلة فلا جديدَ يذكر فيها سوى أنها ازدادت أناقةً.

وبالحديث عن الأناقة فلا يمكننا تجاهل metroToggle، الذي يماثل أدوات checkBox ويعمل عملها.

كما لا يمكننا أن ننسى التطوير الذي جرى على صناديق النصوص، فإضافة خاصية العلامة المائية WaterMark مثلًا تُغني صناديق النصوص وتعطيها لمسة إتقان، وسنأتي على تفصيل صناديق نصوص منصة ميترو أكثر في فقرة قادمة.

وكخاتمة للأدوات الجديدة التي آملُ أن تكون قد ألهمتك وشجعتك لإكمال القراءة والتعرف على المزيد أتركك مع metroTile، والتي هي في الواقع زر button عادي، لكن مسطّح (بالوضع Flat)، وعلاوةً على ذلك تم التحكم بمكان نص الزر.

استخدم الكود التالي:



```
using System;
using System.Windows.Forms;
using MetroFramework;
using MetroFramework.Forms;

namespace MetroTest
{
    public partial class Form1 : MetroForm
    {
        public Form1()
        {
            InitializeComponent();

            // تهيئة اللائحة المنسدلة الأولى لاحتواء أوضاع صندوق الرسالة
            metroComboBox1.Items.Add("Asterisk");
            metroComboBox1.Items.Add("Error");
            metroComboBox1.Items.Add("Exclamation");
            metroComboBox1.Items.Add("Hand");
            metroComboBox1.Items.Add("Information");
            metroComboBox1.Items.Add("None");
            metroComboBox1.Items.Add("Question");
            metroComboBox1.Items.Add("Stop");
            metroComboBox1.Items.Add("Warning");

            metroComboBox1.SelectedIndex = 0;

            // اللائحة المنسدلة الثانية لاحتواء الزر الافتراضي
            metroComboBox2.Items.Add("Button1");
            metroComboBox2.Items.Add("Button2");
            metroComboBox2.Items.Add("Button3");

            metroComboBox2.SelectedIndex = 0;

            // ضبط نصوص التلميحات لبعض الأدوات
            metroToolTip1.SetToolTip(metroToggle1,
                "If on, shows a \"Welcome message\" title, else shows
\"Notification\" title (Default one).");
            metroToolTip1.SetToolTip(metroTextBox1,
                "Your name.");
            metroToolTip1.SetToolTip(metroComboBox1,
                "Determines the icon on MessageBox as a color.");
            metroToolTip1.SetToolTip(metroComboBox2,
                "Determines the default button.");
        }

        private void metroTile1_Click(object sender, System.EventArgs e)
        {
            // مسح سجل مزود الأخطاء
            errorProvider1.Clear();

            // إذا لم يكن نص صندوق النصوص فارغ يتم إظهار رسالة
            if (metroTextBox1.Text != "")
                ShowMessage(metroTextBox1.Text,
                    "Welcome message",
                    MessageBoxButtons.AbortRetryIgnore);
```

```
        else // وإلا يظهر خطأ موضحًا ذلك
            errorProvider1.SetError(metroTextBox1,"You have to enter a name");
    }

    // الإجراءات الخمسة التالية لها نفس وظيفة الإجراء الأخير مع بعض التعديلات
    private void metroTile2_Click(object sender, System.EventArgs e)
    {
        errorProvider1.Clear();
        if (metroTextBox1.Text != "")
            ShowMessage(metroTextBox1.Text,
                "Welcome message",
                MessageBoxButtons.OK);
        else
            errorProvider1.SetError(metroTextBox1,"You have to enter a name");
    }

    private void metroTile3_Click(object sender, System.EventArgs e)
    {
        errorProvider1.Clear();
        if (metroTextBox1.Text != "")
            ShowMessage(metroTextBox1.Text,
                "Welcome message",
                MessageBoxButtons.OKCancel);
        else
            errorProvider1.SetError(metroTextBox1,"You have to enter a name");
    }

    private void metroTile4_Click(object sender, System.EventArgs e)
    {
        errorProvider1.Clear();
        if (metroTextBox1.Text != "")
            ShowMessage(metroTextBox1.Text,
                "Welcome message",
                MessageBoxButtons.RetryCancel);
        else
            errorProvider1.SetError(metroTextBox1,"You have to enter a name");
    }

    private void metroTile5_Click(object sender, System.EventArgs e)
    {
        errorProvider1.Clear();
        if (metroTextBox1.Text != "")
            ShowMessage(metroTextBox1.Text,
                "Welcome message",
                MessageBoxButtons.YesNo);
        else
            errorProvider1.SetError(metroTextBox1,"You have to enter a name");
    }

    private void metroTile6_Click(object sender, System.EventArgs e)
    {
        errorProvider1.Clear();
        if (metroTextBox1.Text != "")
            ShowMessage(metroTextBox1.Text,
                "Welcome message",
                MessageBoxButtons.YesNoCancel);
        else
            errorProvider1.SetError(metroTextBox1,"You have to enter a name");
    }
```

```
        // إجراء لإظهار صندوق الرسالة
        void ShowMessage(string text, string title, MessageBoxButtons buttons)
        {
            //استنساخ معدّد بناء على النص المختار ضمن اللائحة المنسدلة الأولى
            MessageBoxIcon icon = (MessageBoxIcon)Enum.Parse
                (typeof(MessageBoxIcon), metroComboBox1.Text);
            // استنساخ معدد بناء على النص المختار ضمن اللائحة المنسدلة الثانية
            MessageBoxDefaultButton defaultButton =
                (MessageBoxDefaultButton)Enum.Parse
                (typeof(MessageBoxDefaultButton), metroComboBox2.Text);

            // إذا تم اختيار إظهار عنوان، يتم إظهاره
            if (metroToggle1.Checked)
                MetroMessageBox.Show(this,
                    "Welcome mr. " + text,
                    title,
                    buttons,
                    icon,
                    defaultButton);

            else // وإلا فيتم إظهار العنوان الافتراضي لصناديق رسائل منصة ميترو
                MetroMessageBox.Show(this,
                    "Welcome mr. " + text,
                    "Notification",
                    buttons,
                    icon,
                    defaultButton);
        }
    }
}
```

المميز في صناديق رسائل منصة Metro أنها تظهر على عرض النافذة الأم لها (في الكود الأخير استخدمنا الكلمة this للدلالة على النافذة Form1 لتكون نافذةً أم لصندوق الرسالة)، وتظهر بارتفاع معين يمكن تعديله (لم أعدله في الكود)، وبلون يدل على حالة الرسالة (بدلًا من إظهار أيقونة كما في صناديق الرسائل التقليدية يتم إظهار صندوق الرسائل بلون معين، مع إطلاق صوت معبّر عن حالة الرسالة). لاحظ الأمثلة التالية:

Metro MessageBox by Eng27

Welcome message
Welcome mr. Hasan M. al-Fahl

Ok Cancel

Title On

YesNo YesNoCancel

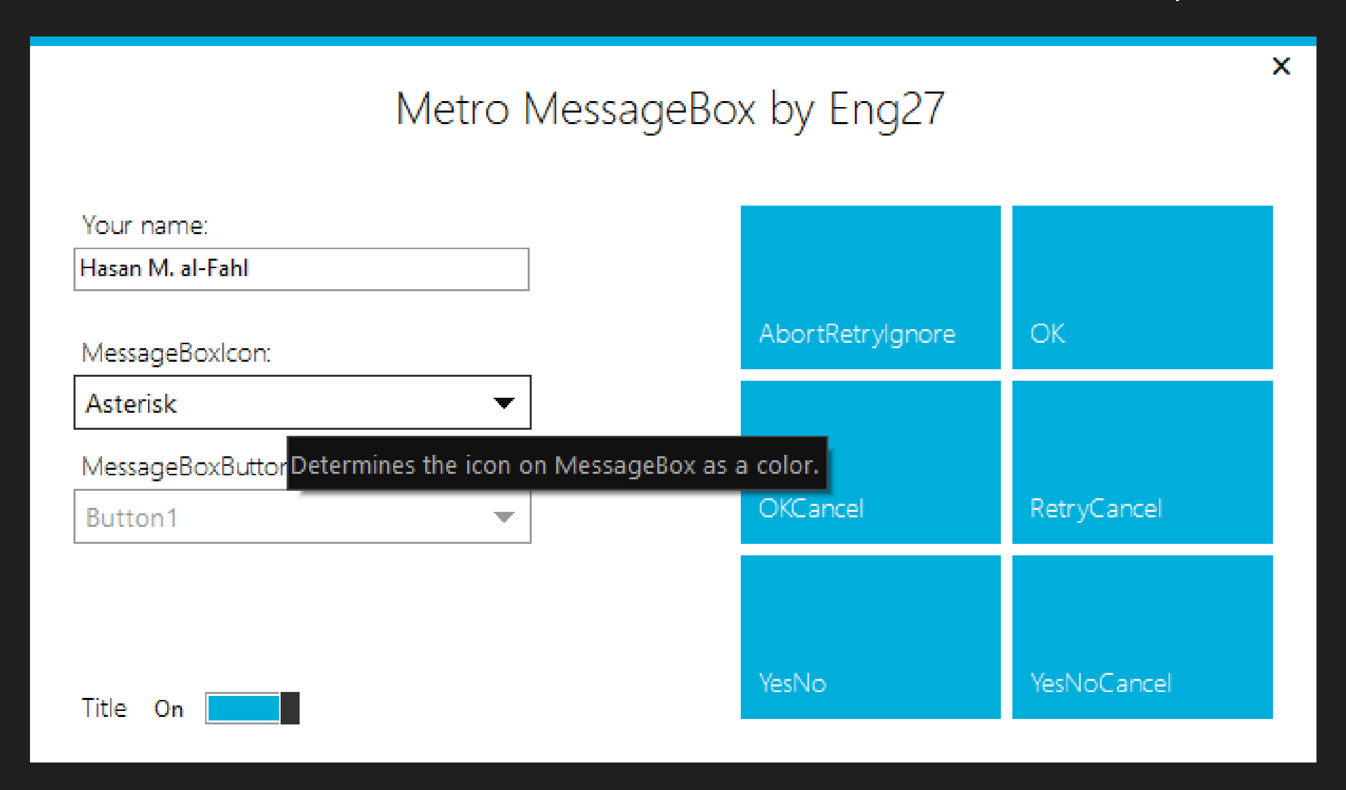
Metro MessageBox by Eng27

Welcome message
Welcome mr. Hasan M. al-Fahl

Yes No

Title On

YesNo YesNoCancel



نافذة تسجيل دخول بسيطة – صناديق النصوص

ذكرنا بعضَ التطويرات على صناديق الرسائل وهنا سنتعمق أكثر بإمكانيات أدوات منصة ميترو، وفي هذا المثال سنتناول أداتين، صناديق النصوص ومدير المظهر.

صمم النافذة التالية:

اضبط خصائص الأدوات المستخدمة كما يلي:

الأداة	الخاصية	القيمة
Form1	AutoScaleMode	None
	MaximizeBox	False
	MinimizeBox	False
	Resizable	False
	ShadowType	AeroShadow
	Size	246 ,428
	Text	Login UI by Eng27
	TextAlign	Center
metroCheckBox1	Text	Remember me
metroTextBox1	DisplayIcon	True
	Size	23 ,222
	ShowButton	True
	WaterMark	Username
metroTextBox2	DisplayIcon	True
	Size	23 ,222
	WaterMark	Password
metroTile1	Text	Login
metroStyleManager1	-	-
pictureBox1	Size	85 ,99

يمكنك من خلال مدير المظهر MetroStyleManager إدارة سِمة نافذتك وأدواتها Theme واللونَ الجانبي Style في تطبيقك.

الجميل في صناديق النصوص في منصة ميترو هو إمكانية إضافة أيقونة وزر للأداة، بالإضافة للعلامة المائية التي سبق ذكرها في المثال الأول من مشاريع ميترو.

يمكنك إضافة أيقونة تظهر على يسار الأداة للدلالة على وظيفتها من خلال الخاصية Icon لضبط الأيقونة، وإضافة زر مع تعيين صورة له من خلال الخاصية ShowButton لتفعيل الزر والخاصية Image ضمن الخاصية CustomButton لضبط خصائص الزر.



لاحظ اللون الأحمر المحيط بالأداة، هذا اللون مضبوط اعتمادًا على الخاصية Style، والتي يمكن ضبطها من خلال الأداة MetroStyleManager ليتم ضبط جميع أدوات النافذة على نفس اللون، وهو اللون الجانبي الذي ذكرته سابقًا.

استخدم الكود التالي:



```
using System;
using System.IO;
using System.Collections.Generic;
using System.Windows.Forms;
using MetroFramework;
using MetroFramework.Forms;

namespace MetroTest
{
    public partial class Form1 : MetroForm
    {
        // لوائح تحوي أسماء المستخدمين وكلمات سرهم
        List<string> users = new List<string>();
        List<string> passwords = new List<string>();
        // متغير يحدد حالة تسجيل الدخول (موَضحة بالمعدّد التالي)
        errorCodes err;
        // معدّد فيه الحالات التي يمكن للبرنامج أن يأخذها أثناء تسجيل الدخول
        enum errorCodes
        {
            None = 100, // أن يكون الوضع تمام
            UsernameOrPasswordNotEntered = 101, // لم يتم تحديد اسم المستخدم أوكلمة المرور
            UsernameOrPasswordAreNotCorrect=102, // اسم المستخدم أو كلمة المرور خاطئة
            UsernameNotFound=103 // اسم المستخدم غير موجود
        }

        public Form1()
        {
            InitializeComponent();

            // تحديد المستخدمين الحاليين
            users.Add("User1");
            users.Add("User2");
            users.Add("User3");
```

```
            // تحديد كلمات سر المستخدمين
            passwords.Add("1234");
            passwords.Add("0000");
            passwords.Add("1111");

            // ضبط مدير المظهر على أنه مدير مظهر النافذة
            metroStyleManager1.Owner = this;
            this.StyleManager = metroStyleManager1;
        }

        // زر تسجيل الدخول
        private void metroTile1_Click(object sender, EventArgs e)
        {
            // إذا لم يتم إدخال اسم المستخدم أو كلمة السر
            if (metroTextBox1.Text == "" || metroTextBox2.Text == "")
            {
                //يتم تحديد حالة البرنامج على أنها 101 (101 هي نفسها عدم إدخال البيانات كما صرّحنا عن ذلك في بنية المعدّد)
                err = (errorCodes)101;
                goto end;
            }
            // إذا لم يكن اسم المستخدم موجودًا
            if (!users.Contains(metroTextBox1.Text))
            {
                // يتم تحديد حالة البرنامج على أنها 103
                err = (errorCodes)103;
                goto end;
            }
            // وإن لم يكن ما سبق (هذا يعني أن اسم المستخدم وكلمة السر مدخلان، واسم المستخدم موجود)
            else //عندها يتم البحث ضمن جميع المستخدمين على كلمة السر من أجل اسم المستخدم المطلوب
                for (int i = 0; i < 3; ++i)
                    if (users[i] == metroTextBox1.Text)
                    {
                        // إذا كانت كلمة السر مطابقة لكلمة سر المستخدم المحدد
                        if (passwords[i] == metroTextBox2.Text)
                            err = errorCodes.None; // يتم ضبط حالة البرنامج على عدم وجود خطأ

                        // وإلا
                        else // يتم تحديد حالة البرنامج على أن كلمة السر أو اسم المستخدم خاطئة
                            err = errorCodes.UsernameOrPasswordAreNotCorrect;
                        break; // إنهاء الإجراء حتى لا يتم المرور من العنوان التالي (بعد 3 أسطر)
                    }

            // عنوان يتم الانتقال إليه عند حالات معينة
            end:
            // إذا لم يكن هناك خطأ
            if (err == errorCodes.None)
            {
                MetroMessageBox.Show(this,
                    "Login done successfully!",
                    "Login success!",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Information,
                    150);

                metroStyleManager1.Style = MetroColorStyle.Green;

                // هنا يجب استدعاء طريق تقوم بتسجيل الدخول والانتقال للنافذة الرئيسية للبرنامج
            }
```

```
            // في حال كان اسم المستخدم أو كلمة السر غير مدخلَين
            else if (err == errorCodes.UsernameOrPasswordNotEntered)
            {
                MetroMessageBox.Show(this,
                    "You must enter username and password!",
                    "Error on login!",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error,
                    150);

                metroStyleManager1.Style = MetroColorStyle.Red;
            }

            // في حال كانت كلمة السر غير صحيحة
            else if (err == errorCodes.UsernameOrPasswordAreNotCorrect)
            {
                MetroMessageBox.Show(this,
                    "Password or username are incorrect!",
                    "Error on login!",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error,
                    150);

                metroStyleManager1.Style  = MetroColorStyle.Red;
            }

            // في حال لم يكن هناك مستخدم بهذا الاسم
            else if (err == errorCodes.UsernameNotFound)
            {
                MetroMessageBox.Show(this,
                    "No such username!",
                    "Error on login!",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error,
                    150);

                metroStyleManager1.Style = MetroColorStyle.Red;
            }
        }

        private void metroTextBox1_ButtonClick(object sender, EventArgs e)
        {
            // يفتح نافذة إنشاء مستخدم جديد
        }
    }
}
```

تعمّدت استخدام المعدّدات Enumerations بشكلين، الأول بذكر المعدد كاملًا، والثاني بالتعامل معه على شكل أعداد ثم تحويلها لمعدد. كما حاولت تقريب فكرة الأخطاء البرمجية 404 وغيرها في هذا المعدد، لتستخدمها ضمن برامجك إن أحببت ذلك.

قائمة مهام بسيطة

يمكنك عرض قائمة مهام باستخدام الأداة listView أو الأداة gridView، ويمكنك تخزين البيانات في قاعدة بيانات أو ضمن ملفات من أنواع معينة، أو تخزين كل عنصر بملف منفصل. أيًّا كانت طريقة التخزين المتبعة فإن مثال هذه الفقرة سيغض النظر عنها ويعطي الأهمية لأسلوب التصميم باستخدام منصة Metro.

صمم الواجهة التالية:

اضبط الخصائص كما يلي:

الأداة	الخاصية	القيمة
	AutoScaleMode	None
Form1	MaximizeBox	False
	MinimizeBox	False

	Resizable	False
	ShadowType	AeroShadow
	Size	425 ,641
	Text	myTODO by Eng27
metroTextBox1	Size	23 ,269
	WaterMark	Todo caption
metroTextBox2	Size	23 ,269
	WaterMark	Todo detail
metroButton1	Text	Add
metroButton2	Text	Update
metroButton3	Text	Delete
metroButton4	Text	Delete all

استخدم الكود التالي:



```
using System;
using System.Windows.Forms;
using MetroFramework.Forms;

namespace MetroTest
{
    public partial class Form1 : MetroForm
    {
        bool isSelected; // متغير يحدد فيما إذا كان أحد عناصر اللائحة محدد

        public Form1()
        {
            InitializeComponent();
        }

        private void metroListView1_ItemSelectionChanged
             (object sender, ListViewItemSelectionChangedEventArgs e)
        {
          if (e.IsSelected) {
            metroTextBox1.Text = metroListView1.SelectedItems[0].SubItems[0].Text;
            metroTextBox2.Text = metroListView1.SelectedItems[0].SubItems[1].Text;
            isSelected = true; }
          else
            isSelected = false;
        }
```

```
        private void metroButton1_Click(object sender, EventArgs e)
        {
            if (!string.IsNullOrWhiteSpace(metroTextBox1.Text))
            {
                metroListView1.Items.Add(
                    new ListViewItem(
                        new[] { metroTextBox1.Text, metroTextBox2.Text }));
                metroTextBox1.Clear();
                metroTextBox2.Clear();
            }
        }

        private void metroButton2_Click(object sender, EventArgs e)
        {
            if (isSelected) // لا يمكن التعديل إلا إذا كان هناك عنصر محدد
            {
                int selectedIndex = metroListView1.SelectedIndices[0];
                metroListView1.Items.RemoveAt(selectedIndex);
                metroListView1.Items.Insert(selectedIndex, new ListViewItem(
                    new[] { metroTextBox1.Text, metroTextBox2.Text }));
                metroTextBox1.Clear();
                metroTextBox2.Clear();
            }
        }

        private void metroButton3_Click(object sender, EventArgs e)
        {
            if (isSelected) // لا يمكن الحذف إلا إذا كان هناك عنصر محدد
            {
                int selectedIndex = metroListView1.SelectedIndices[0];
                metroListView1.Items.RemoveAt(selectedIndex);
                metroTextBox1.Clear();
                metroTextBox2.Clear();
            }
        }

        private void metroButton4_Click(object sender, EventArgs e)
        {
            metroListView1.Items.Clear();
            metroTextBox1.Clear();
            metroTextBox2.Clear();
        }
    }
}
```

يمكنك الحصول على المزيد من قناة مبرمج المنصة DENRIC DENISE - INFO [1]، كما يمكنك الاطلاع على موقعه من هنا [2].

[1] https://www.youtube.com/channel/UCs3-k18JID-4PyCl1SJt9Vw.
[2] http://denricdenise.info.

منصة XanderUI

منصة خفيفة، جميلة، بأدوات كثيرة ومسطحة Flat. هي منصة غير مشهورة، ومع هذا فلا بد من تجربتها والاستفادة منها. يمكنك تحميلها من صفحة مبرمج المنصة على GitHub Ricky Divjakovski [1]، كما يمكنك الاشتراك بقناته على اليوتيوب Ricky's Tutorials [2] لمتابعة فيديوهاته وأمثلته.

لا تنسَ إضافة أدوات هذه المنصة إلى صندوق الأدوات.

تعطيك هذه المنصة الأدوات التالية:

- FormDropShadow
- XUIBackgroundSleeper
- XUIBanner
- XUIBarGraph
- XUIBatteryPercentageAPI
- XUIButton
- XUICard
- XUICheckBox
- XUICircleProgressBar
- XUIClock
- XUIColorPane
- XUIColorPicker
- XUICustomGroupbox
- XUICustomPictureBox
- XUICustomToolstrip
- XUIExtendedFSWatcher
- XUIFlatMenuStrip
- XUIFlatProgressBar
- XUIFlatTab
- XUIFormDesign
- XUIFormHandle
- XUIGauge
- XUIGradientPanel
- XUIJoyStick
- XUILineGraph
- XUINavigationBar
- XUIObjectAnimator
- XUIObjectEllipse
- XUIPieGraph
- XUIRadio
- XUISegment
- XUISlider
- XUISlidingPanel
- XUISplashScreen
- XUISuperButton
- XUISwitch
- XUIVolumeController
- XUIWeatherClient
- XUIWidgetPanel
- XUIWifiPercentageAPI

[1] https://github.com/Ricky310711/XanderUI.
[2] https://www.youtube.com/user/RickySpyte.

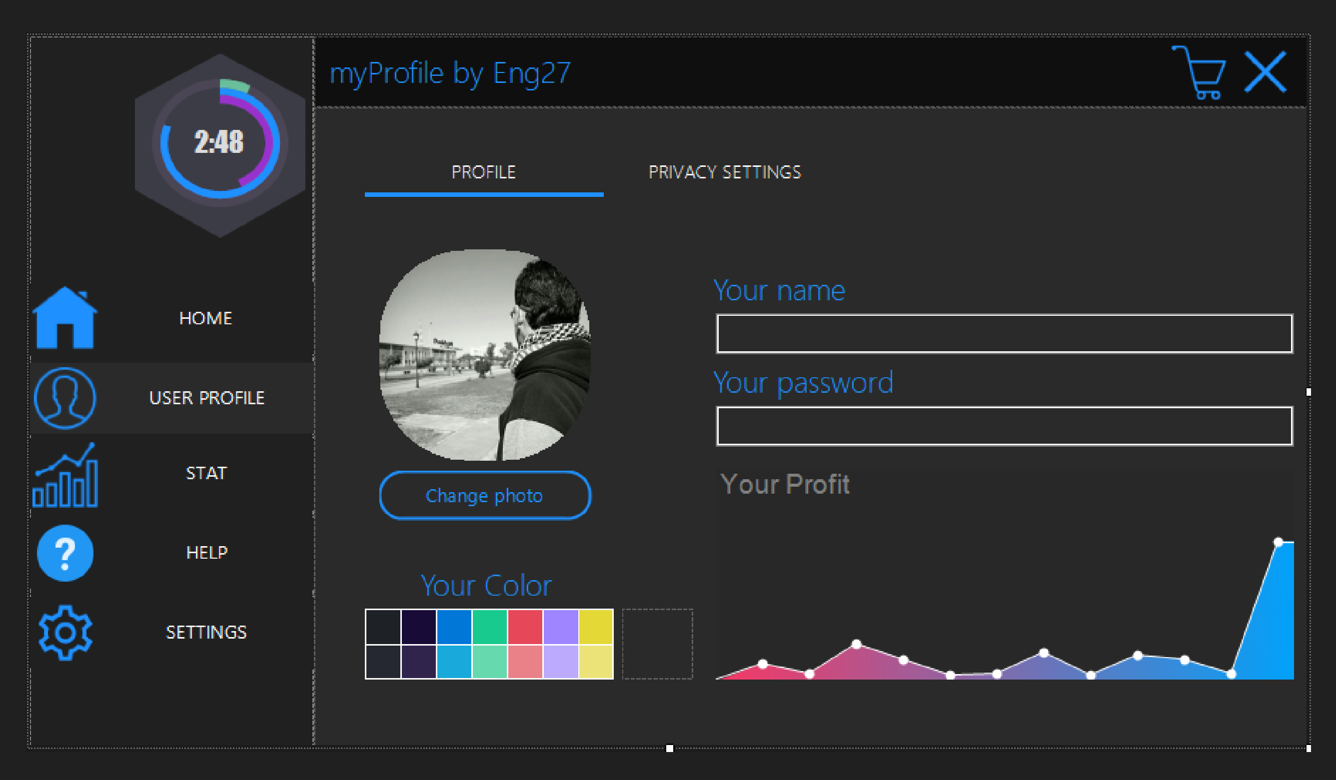
لا يسعنا بالتأكيد ذكر الأدوات كلها، فبعضها مكرر [1] وبعضها عادي جدًّا ولا يختلف عن الأدوات القياسية كثيرًا.. الجدير ذكره هو وجود بعض الأدوات التي تعمل كـ API، فيمكنك من خلالها الحصول على بعض الأمور، مثل معلومات عن شبكة الـ WiFi ومعلومات عن البطارية ومعلومات عن الطقس والتحكم بالصوت.. المنصة بالأساس موجهة للواجهات الليلية، لكن هذا لا يعني إمكانية استخدامها بألوان غير داكنة.

قبل الخوض في مشاريع هذه المنصة لنتحدث عن أدواتها بإيجاز. للمنصة أداة تعطي برنامجك شاشة ترحيب SplashScreen وهي ولو أنها ليست قوية إلا أن فكرة وجودها ضمن المنصة كافية لإغنائها، كما تعطيك المنصة أدوات مسطحة عصرية FlatModernControls لعل أكثر ما قد يلفت انتباهك من بينها أدوات البيانات، والأدوات الدائرية.. ومن الإضافات الجميلة في هذه المنصة وجود أدوات مثل اللائحة المنزلقة SlidingPanel وأداة Segmant (التي تضيف إلى برنامجك تبويبات على شكل أقسام). وأخيرًا، لا بد من ذكر أداة تدوير الحواف ObjectEllipse وأداة التقاط النافذة FormHandle وأداة المحاكاة ObjectAnimator والتي لا غنى عنها في إحياء النوافذ.

نافذة ملف شخصي بسيطة

يمكنك – على سبيل المثال لا الحصر – من خلال منصة XanderUI تصميم نافذة ملف شخصي كما يلي:

[1] التكرار ليس عيبًا في المنصات، لا بل على العكس فالتكرار ولو بمزايا قريبة جدًّا من الأدوات التقليدية يجعل اعتماد المستخدم على المنصة وعدم استخدامه للمنصات الأخرى أكبر، فلو أنه في منصة ما أداة زر بشكل وسلوك معينين وأردت استخدام الشكل التقليدي للزر أو أحد مزاياه فإنك ستذهب لمنصة أخرى، في حين لو أن المنصة التي تستخدمها تدعم ضمن مزايا أدواتها الأشكال القياسية خصوصًا مع الأدوات الأكثر استخدامًا فإنك لن تستخدم غيرها!!



الفكرة من هذه النافذة إظهار بيانات ملف شخصي للمستخدمين، فيها صورة المستخدم واللون الخاص به واسمه وكلمة سره، ومعدل أرباحه (كل هذه الأمور افتراضية ويمكن الإضافة عليها أو تعديلها).

النافذة فيها على اليسار قائمة بأقسام البرنامج، والتي لن نناقش أيّا منها في هذا المثال عدا قائمة USER PROFILE، وفيها من الأعلى اسم البرنامج وبعض الأزرار، وفي الوسط الملف الشخصي مجزّأ إلى قسمين، حيث سنناقش تصميم الجزء PROFILE فقط.

استخدمنا أدوات مختلفة من منصة XanderUI، وبعض أدوات Windows القياسية. هناك ثلاثة أدوات لا تظهر ضمن المصمم، هي أداة XUIObjectEllipse وأداتين من النوع UXIFormHandle، سيتم إيضاحها في جدول قادم.

قم بوضع أداة XUISliderPanel أولًا، ثم أداة Panel لها الخاصية Dock = Top، ثم الأداة Panel لها الخاصية Dock = Fill (تأكد من أن الأداة الأخيرة هي ضمن أعلى طبقة من طبقات الأدوات، فإن لم تكن كذلك انقر عليها باليمين ثم Send to Front).

الأدوات المستخدمة موضحة بالجدول التالي:

الأداة	الخاصية	القيمة
Form1	AutoScaleMode	None
	Font	Segoe UI, 9.75pt
	FontBorderStyle	None
	Size	900, 500
button1 & button2	Anchor	Top, Right
	BackColor	15, 15, 15
	BackgroundImageLayout	Zoom
	FlatAppearance.BorderSize	0
	FlatStyle	Flat
	Size	40, 40
	هاتين الأداتين موجودين في panel1	
label1	Font	Segoe UI Light, 15.75pt
	ForeColor	DodgerBlue
	Text	Your Color
label2	Font	Segoe UI Light, 15.75pt
	ForeColor	DodgerBlue
	Text	myProfile by Eng27
label3	Font	Segoe UI Light, 15.75pt
	ForeColor	DodgerBlue
	Text	Your name



Segoe UI Light, 15.75pt	Font	label4
DodgerBlue	ForeColor	
Your password	Text	
Top	Dock	panel1
200, 0	Location	
700, 50	Size	
Fill	Dock	panel2
200, 50	Location	
217, 353	Location	panel3
50, 50	Size	
Top, Left, Right	Anchor	textBox1
Top, Left, Right	Anchor	textBox2
True	UseSystemPasswordChar	
42, 42, 42	BackgroundColor	xuiButton1
Invert	ButtonStyle	
Change photo	ButtonText	
DodgerBlue	ClickBackColor	
DodgerBlue	ClickTextColor	
DodgerBlue	HoverBackgroundColor	
42, 42, 42	HoverTextColor	
32, 32, 32	BackColor	xuiClock1
175, 50	Size	xuiColorPane1
42, 42, 42	BackColor	xuiCustomPictureBox1
150, 150	Size	

panel1	HandleControl	xuiFormHandle1
panel2	HandleControl	xuiFormHandle2
Top, Left, Right	BackColor	xuiLineGraph1
Your Profit	GraphTitle	
125	CornerRadius	xuiObjectEllipse1
xuiCustomPictureBox1	EffectedControl	
42, 42, 42	BackColor	xuiSegment1
PROFILE, PRIVACY SETTINGS	Items	
White	SegmentActiveTextColor	
42, 42, 42	SegmentBackColor	
DodgerBlue	SegmentColor	
White	SegmentInactiveTextColor	
Material	SegmentStyle	
32, 32, 32	BottomLeft	xuiSlidingPanel1
32, 32, 32	BottomRight	
True	Collapsed	
50	PanelWidthCollapsed	
200	PanelWidthExpanded	
200, 50	Size	
32, 32, 32	PrimerColor	
Horizontal	Style	
32, 32, 32	TopLeft	
32, 32, 32	TopRight	
32, 32, 32	BackgroundColor	

xuiSuperButton1 [1]	ButtonStyle	Flat
	ButtonText	HOME
	HoverBackgroundColor	52, 52, 52
	HoverTextColor	DodgerBlue
	SelectedBackColor	42, 42, 42
	SelectedTextColor	White
	TextColor	White

للتحكم بالنافذة بشكل مبدئي، استخدم الكود التالي:



```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace XanderTest {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
            xuiObjectEllipse1.EffectedControl = xuiCustomPictureBox1;
            //من المفترض أن هذه القيم يتم أخذها من قاعدة بيانات البرنامج، ولكن ما يهمنا هو التصميم وليس التكويد
            xuiColorPane1.SelectedColor = Color.DodgerBlue;
            panel3.BackColor = Color.DodgerBlue;
        }

        private void xuiColorPane1_Click(object sender, EventArgs e) {
            //إظهار اللون المختار
            panel3.BackColor = xuiColorPane1.SelectedColor;
        }

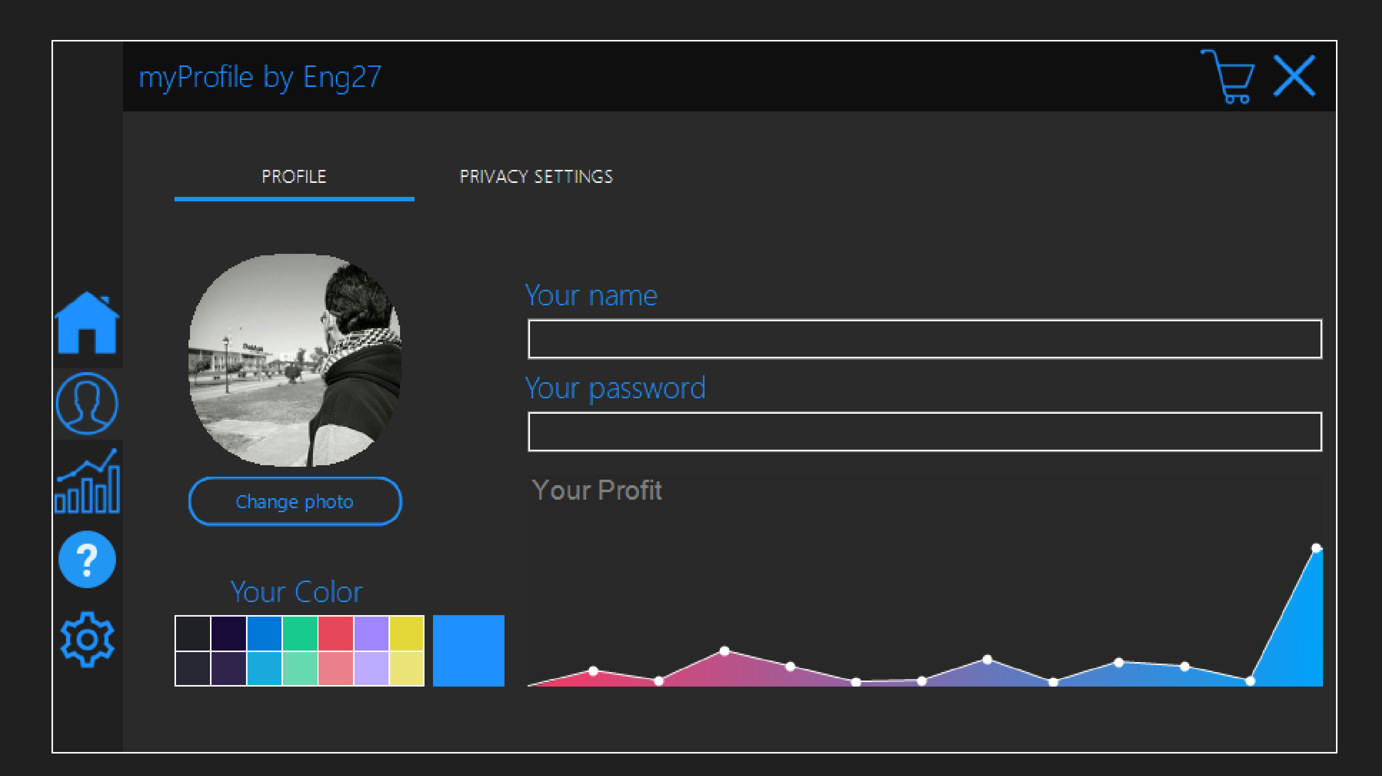
        private void xuiSlidingPanel1_MouseEnter(object sender, EventArgs e) {
            //عند دخول الفأرة نطاق اللائحة اليسرى يتم فتحها
            xuiSlidingPanel1.Collapsed = false;
        }

        private void panel1_MouseEnter(object sender, EventArgs e) {
            //عند دخول الفأرة نطاق اللاحئة العليا يتم إغلاق اللائحة اليسرى
            xuiSlidingPanel1.Collapsed = true;
        }

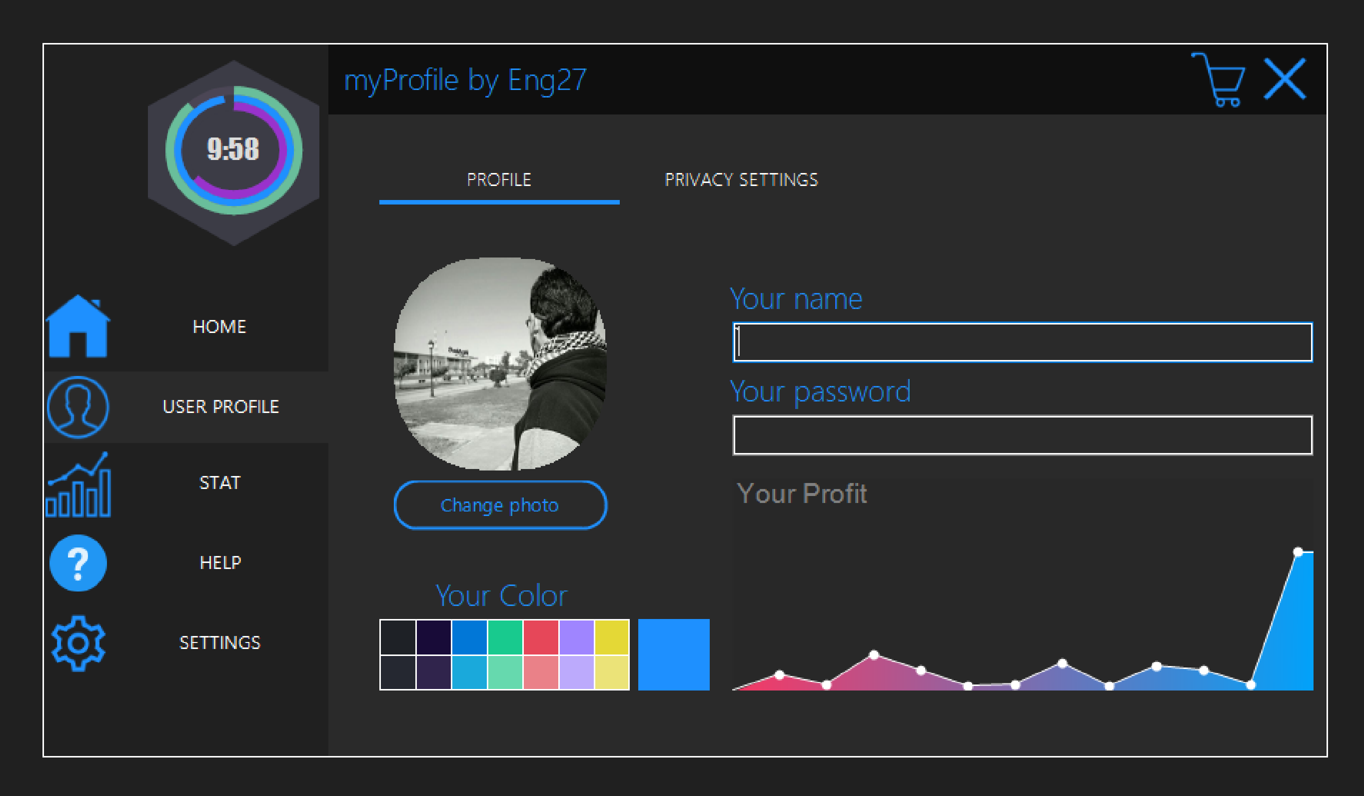
        private void panel2_MouseEnter(object sender, EventArgs e) {
            //عند دخول الفأرة نطاق اللائحة الوسطى يتم إغلاق اللائحة اليسرى
            xuiSlidingPanel1.Collapsed = true;
        }
    }
}
```

[1] هناك أربع أدوات أخرى من نفس نوع هذه الأداة وتحمل نفس خصائص الأداة عدا الخاصية ButtonText.

عند تشغيل البرنامج، ستحصل على ما يلي:



عند دخول الفأرة في اللائحة اليسرى:



للمزيد، أحيلك إلى مجموعة من التصاميم [1].

منصة Guna.UI

مع الأسف فهذه المنصة غير مشهورة، ومعرفتي للمنصة كان أشبه بصدفة غريبة جدًّا. يمكنك تحميل المنصة من موقع GitHub من هنا [2]. في الفترة التي كتبت فيها هذا الفصل لم تكن هناك إلا النسخة Guna.UI V1، والتي استخدمتها لإنشاء تصاميم هذه الفقرة، وقبيل نشر الكتاب بفترة بسيطة تم طرح النسخة Guna.UI V2 منها.

وقبل أن نبدأ، أحب أن أحيلك إلى بعض الروابط التي قد تنفعك في مشوارك مع هذه المنصة [3].

---

[1] راجع هذه الروابط:
تصميم 1:
https://medium.com/bunifuframework/modern-admin-panel-crafting-utilizing-bunifu-ui-framework-b737bf5bcb5a
تصميم 2:
https://medium.com/bunifuframework/a-definitive-guide-for-creating-a-booking-dashboard-b4402fc53b66
تصميم 3:
https://medium.com/bunifuframework/step-by-step-redesigning-agilecrms-home-ui-dashboard-7d843e0011b7

[2] رابط تحميل منصة Guna.UI https://github.com/sobatdata/Guna.UI-Framework-Lib.
[3] قناة صاحب المنصة على اليوتيوب https://www.youtube.com/channel/UC8b2EWrFfWk8vcgH496Vygw.
روابط فيها مشاريع وأمثلة على نفس المنصة:
https://github.com/sobatdata?tab=repositories
https://www.youtube.com/playlist?list=PLvlJ3T_X1cWW9W1n7pN9QuITPLDV_CPZU
https://www.youtube.com/channel/UClbUBqDWE3kzIHaAQ8IOYuQ/featured

كالعادة، لا تنسَ إضافة أدوات المنصة إلى أدوات الفيجوال ستوديو ضمن قائمة خاصة، والتي هي كما يلي:

GunaAdvenceButton
GunaAdvenceTileButton
GunaAnimateWindow
GunaButton
GunaCheckBox
GunaCircleButton
GunaCirclePictureBox
GunaCircleProgressBar
GunaColorTransition
GunaComboBox
GunaContextMenuStrip
GunaControlBox
GunaDataGridView
GunaDateTimePicker
GunaDragControl
GunaElipse
GunaElipsePanel
GunaGauge
GunaGoogleSwitch
GunaGradient2Panel
GunaGradientButton
GunaGradientCircleButton
GunaGradientPanel
GunaGradientTileButton
GunaGroupBox
GunaHScrollBar
GunaImageButton
GunaImageCheckBox
GunaImageRadioButton
GunaImageReplaceColor

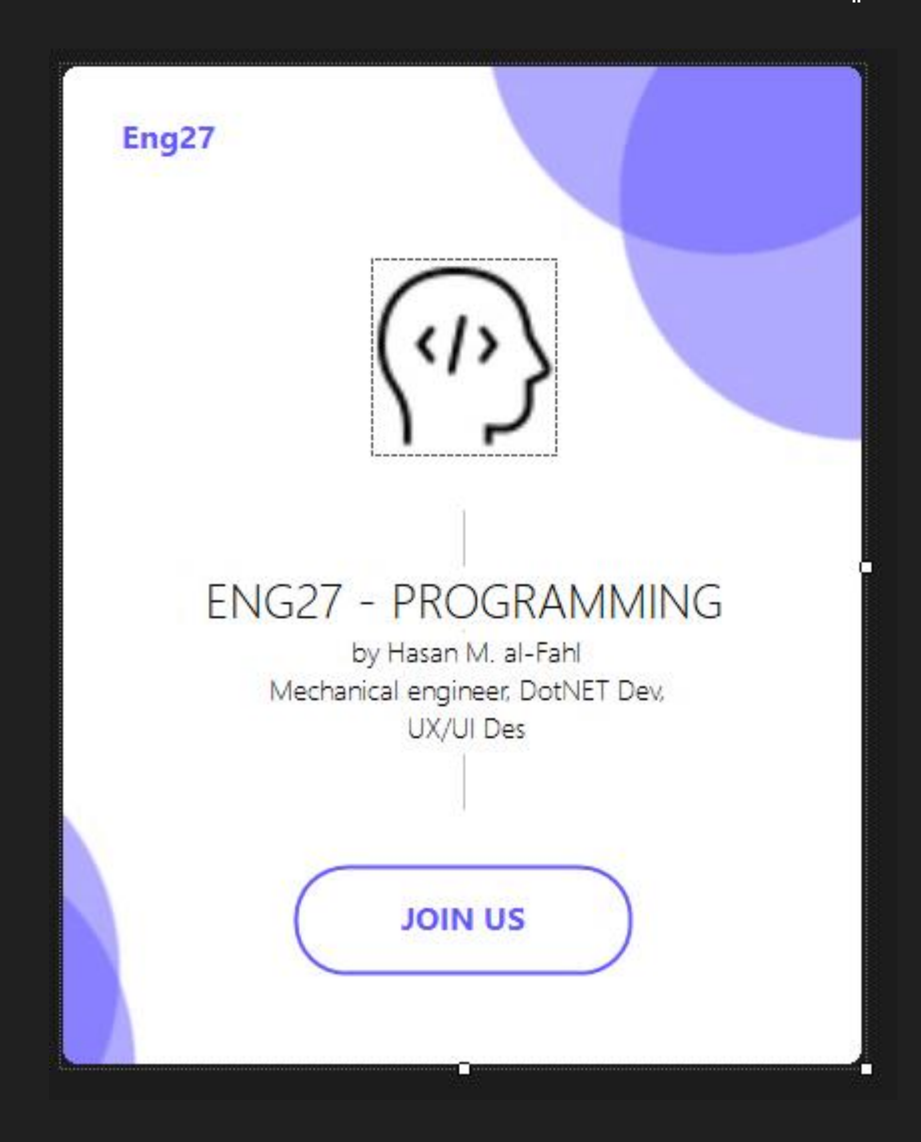
GunaLabel
GunaLinePanel
GunaLineTextBox
GunaLinkLabel
GunaMediumCheckBox
GunaMediumRadioButton
GunaMetroTrackBar
GunaMetroVTrackBar
GunaMouseStateHelper
GunaNumeric
GunaPanel
GunaPictureBox
GunaProgressBar
GunaRadioButton
GunaResize
GunaResizeControl
GunaSeparator
GunaShadowPanel
GunaSwitch
GunaTextBox
GunaTileButton
GunaTrackBar
GunaTransfarantPictureBox
GunaTransition
GunaVProgressBar
GunaVScrollBar
GunaVSeparator
GunaVTrackBar
GunaWinCircleProgressIndicator
GunaWinSwitch

لاحظ التنوع الواسع في الأدوات!!

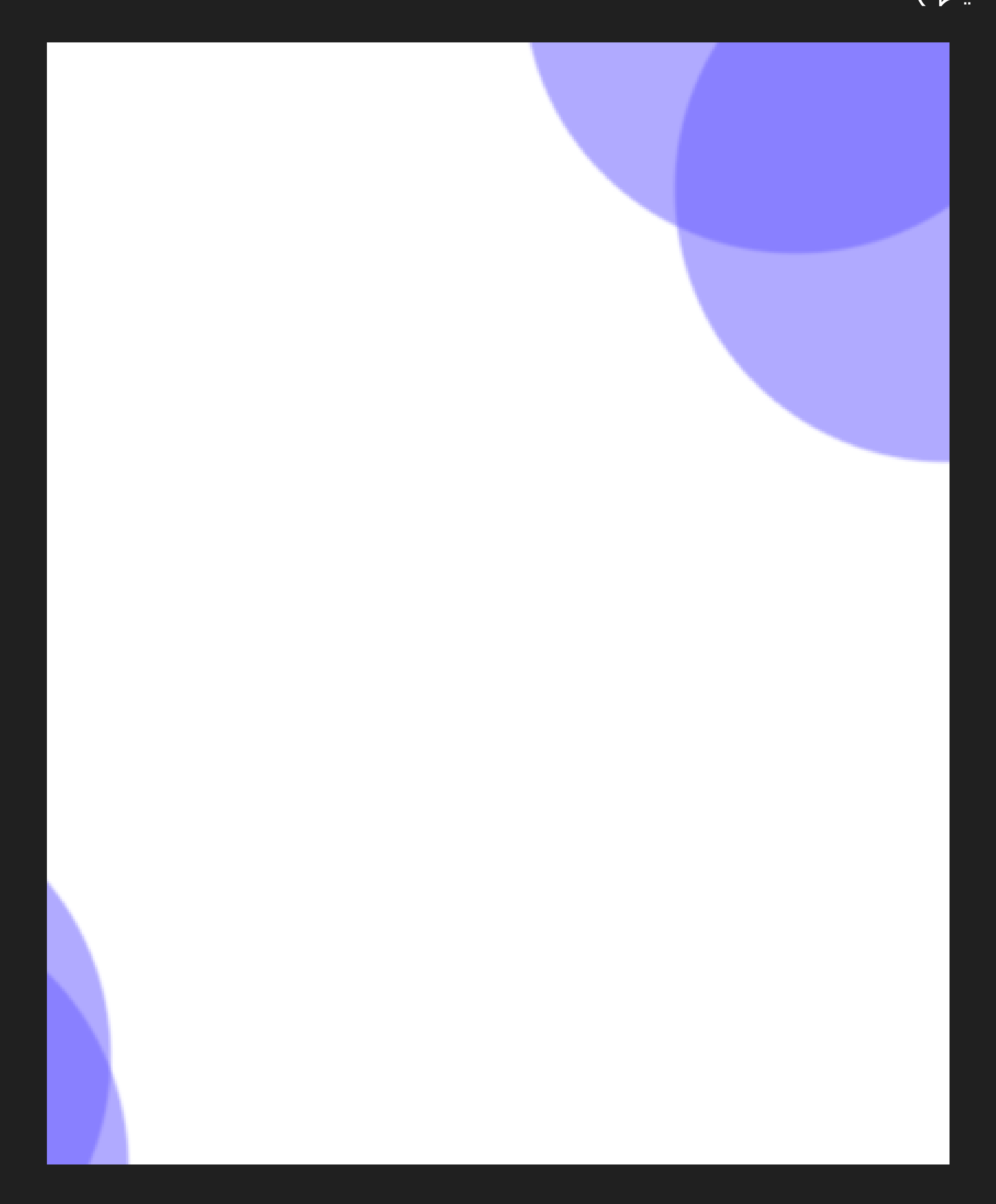
بداية سريعة – نافذة إعلانية صغيرة

خير ما نبدأ به هو المثال المرفق مع ملف تحميل المنصة، وهو لا يزيد عن نافذة صغيرة الحجم فيها زر أسفل منتصفها ذو تأثيرات رسومية عند حركة المؤشر عليه..

صمم النافذة التالية:



استخدم الصورة المستخدمة في المثال الأصلي (يمكنك اقتطاعها من هنا إذا لم تستطع تحصيلها):

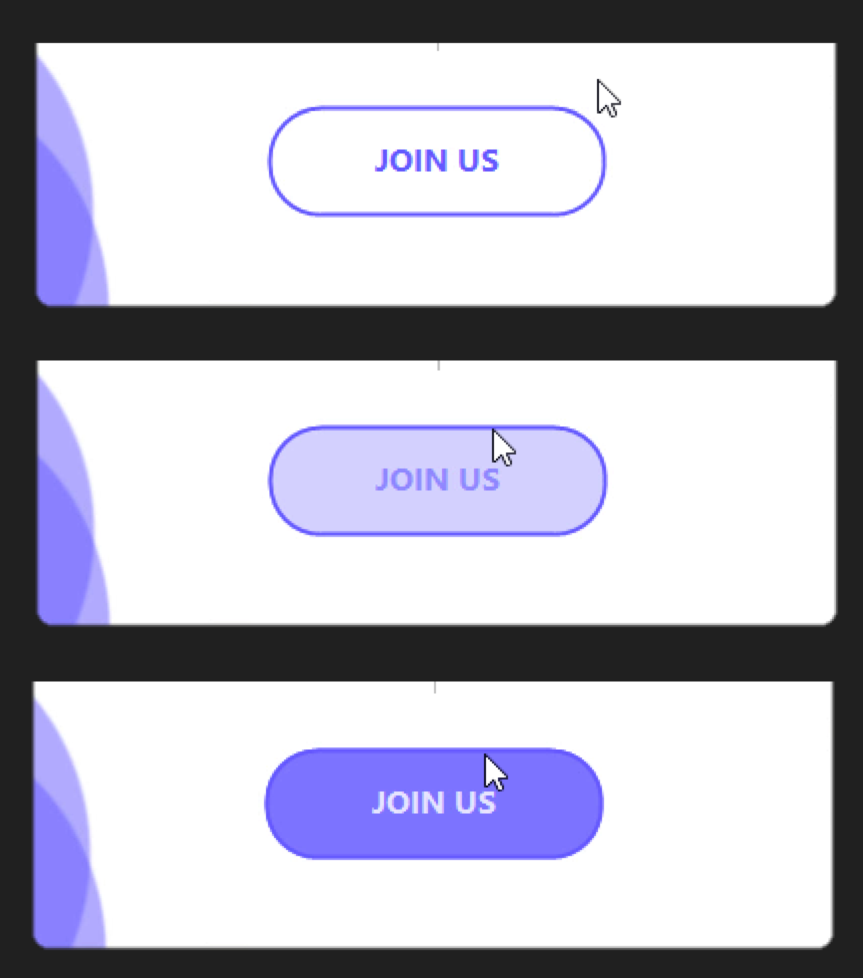


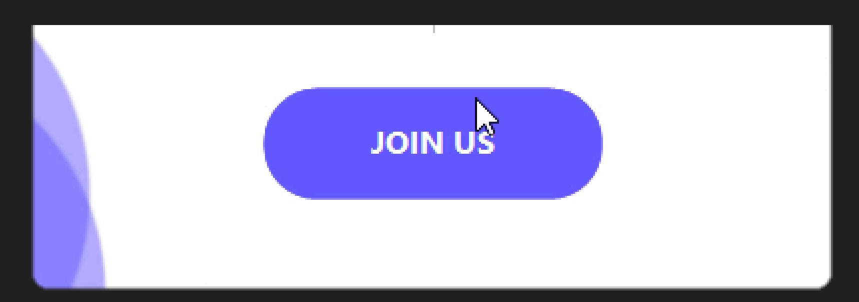
الجدول التالي فيه خصائص أدوات المثال:

الأداة	الخاصية	القيمة
Form1	AutoScaleMode	None
	Font	Segoe UI, 12pt
	FontBorderStyle	None
	Size	400, 500
	Start Position	CenterScreen
gunaButton1	Animated	True
	BaseColor	White
	BorderColor	100, 88, 255
	BorderSize	2
	ForeColor	100, 88, 255
	Image	None
	Location	115, 399
	OnHoverBaseColor	100, 88, 255
	OnHoverBorderColor	100, 88, 255
	OnHoverForeColor	White
	OnPressedColor	Black
	Radius	26
	Size	170, 56
	Text	JOIN US
	TextAlign	Center
gunaDragControl1	TargetControl	Form1
gunaElipse1	Radius	5
	TargetControl	Form1

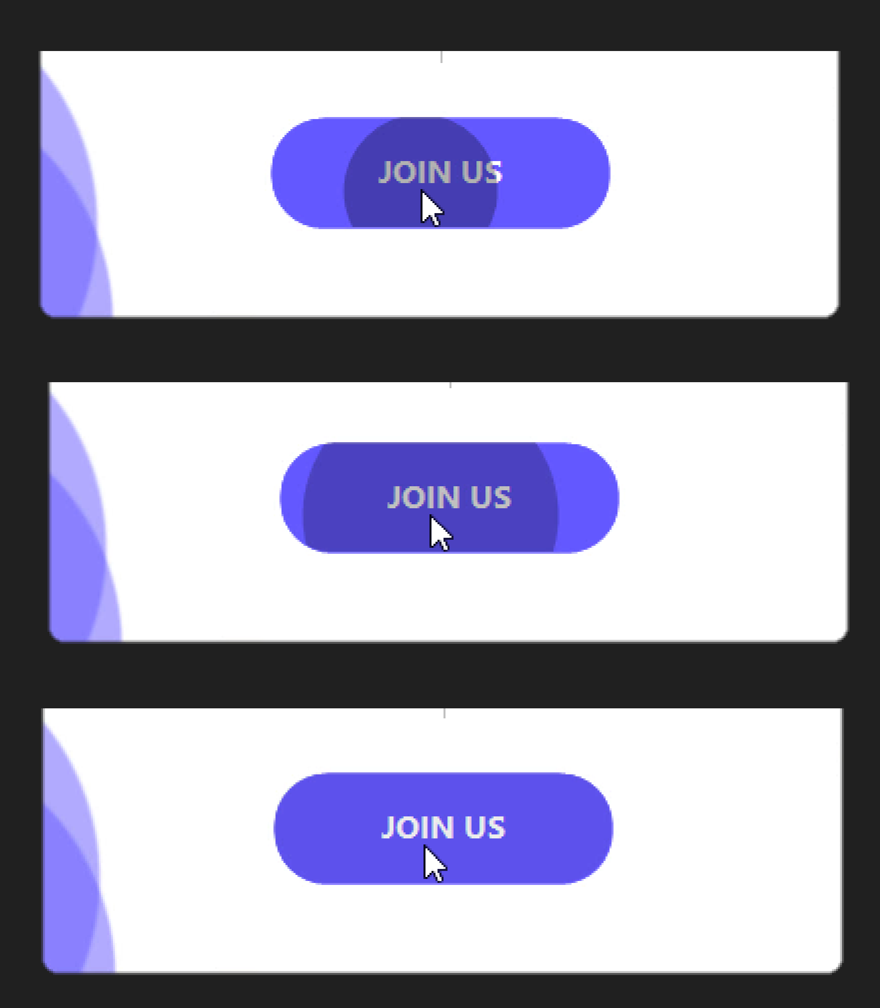
Segoe UI, 12pt, style=Bold	Font	gunaLabel1
Segoe UI Light, 18pt	Font	gunaLabel2
Segoe UI Light, 10pt	Font	gunaLabel3
154, 96	Location	gunaPictureBox1
Zoom	SizeMode	
93, 99	Size	
195, 222	Location	gunaVSeparator1
10, 150	Size	

عند تشغيل المثال ومرور مؤشر الفأرة على الزر، ستحصل على:





وعند النقر عليه:

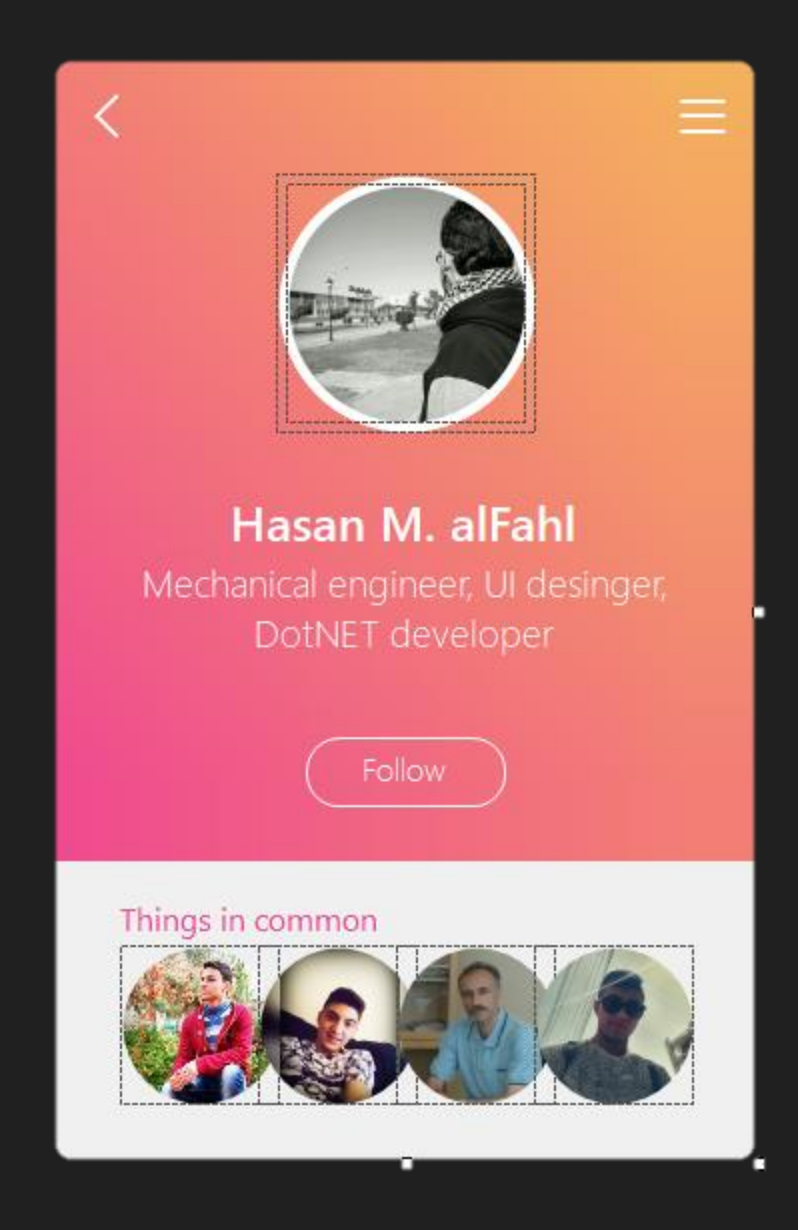


وهذا – رغم بساطته – يُظهر قوة وجمال منصة Guna.UI، وسحرها وإبداع أدواتها.. فالرسوم المتحركة Animations تعني الكثير عزيزي القارئ، الرسوم المتحركة تجعل نوافذ تطبيقاتك حية!! وكما يقول المثل: أطعم العين يستحي الفم.. وعليه، كلما أطعمت أعين مستخدمي تطبيقاتك كلما قل نقّهم[1]!

[1] النق: الشكوى.

نافذة ملف شخصي صغيرة، منبثقة

صمم النافذة التالية:



اضبط الخصائص كما يلي:

الأداة	الخاصية	القيمة
Form1	AutoScaleMode	None
	Font	Segoe UI, 12pt
	FontBorderStyle	None
	Size	350, 550
	Start Position	CenterScreen
gunaCirclePictureBox1	BackColor	Transparent
	BaseColor	Transparent
	Size	120, 120
	SizeMode	StretchImage
	UseTransfarantBackground	True
	هذه الأداة التي تحوي الصورة في الأعلى	
gunaCirclePictureBox2	BackColor	Transparent
	BaseColor	White
	Size	130, 130
	هذه الأداة تحت الأداة السابقة	
gunaCirclePictureBox3	BackColor	Transparent
	BaseColor	Transparent
	Location	32, 442
	Size	80, 80
	SizeMode	StretchImage
	هناك ثلاث أدوات أخرى بنفس مواصفات هذه الأداة إلا بالموقع، حيث تأخذ القيمة Left لها على الترتيب القيم : 101، 170، 239	



True	Animated	gunaButton1
Transparent	BaseColor	
Transparent	BaseColor	
White	BorderColor	
1	BorderSize	
White	ForeColor	
None	Image	
239, 69, 146	OnHoverBaseColor	
White	OnHoverBorderColor	
White	OnHoverForeColor	
239, 69, 146	OnPressedColor	
10	OnPressedDepth	
16	Radius	
100, 36	Size	
Follow	Text	
Center	TextAlign	
Form1	TargetControl	gunaDragControl1
gunaGradientPanel1	TargetControl	gunaDragControl1
5	Radius	gunaElipse1
Form1	TargetControl	
Top	Dock	gunaGradientPanel1
239, 69, 146	GradiantColor1	
239, 69, 146	GradiantColor2	
245, 184, 90	GradiantColor3	
245, 184, 90	GradiantColor4	

	Height	400
gunaImageButton1 & gunaImageButton2	BackColor	Transparent
	BackgroundImageLayout	Zoom
	Size	30, 30
gunaLabel1	Font	Segoe UI Semibold, 18pt, style=Bold
gunaLabel2	Font	Segoe UI Light, 14.25pt
gunaLabel3	Font	Segoe UI, 12pt
gunaVSeparator1	Location	195, 222
	Size	10, 150

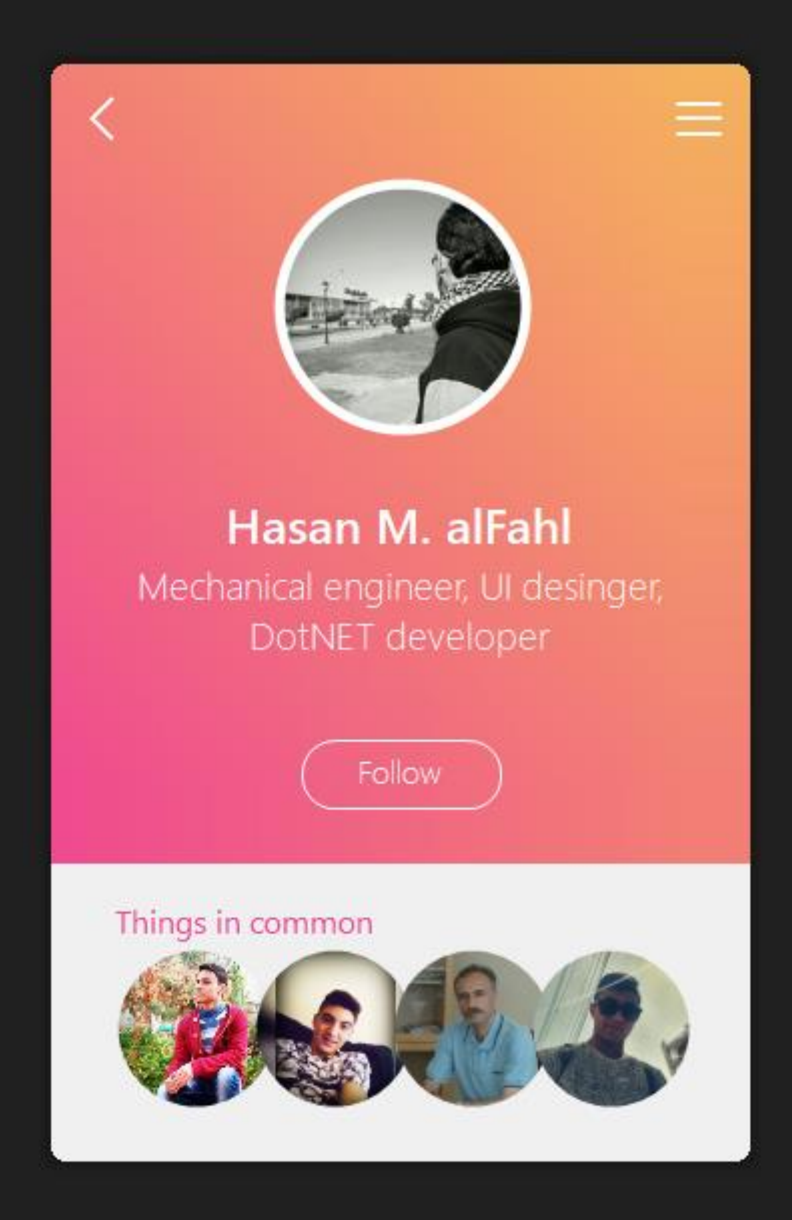
استخدم الكود التالي:



```
using Guna.UI.Lib;
using System.Drawing;
using System.Windows.Forms;

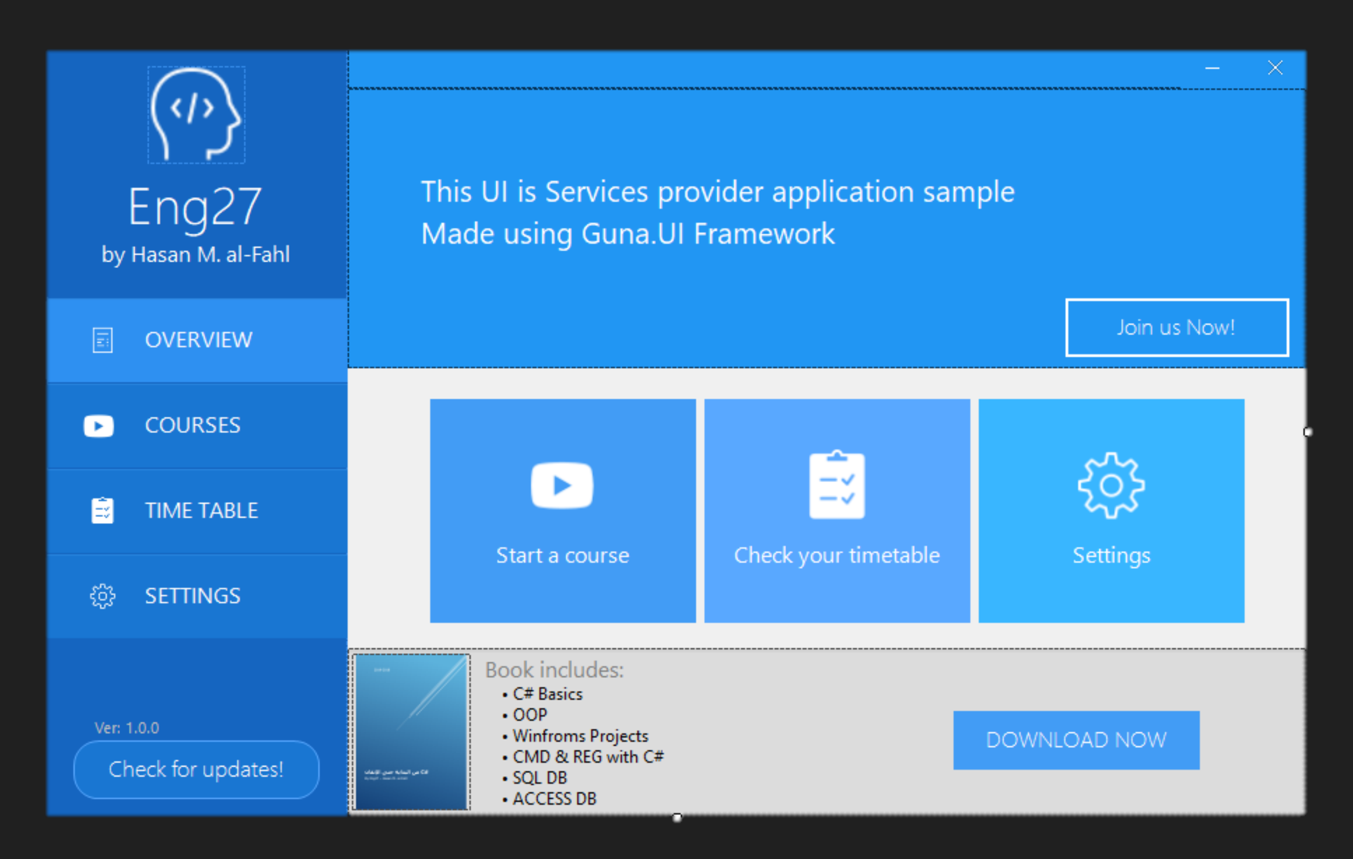
namespace GunaUITest2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            GraphicsHelper.ShadowForm(this);
        }
    }
}
```

شغّل البرنامج:



برنامج عرض خدمات

صمم النافذة التالية (قبل ذلك، انتقل لنهاية المشروع ولاحظ النتيجة، ثم عد):



أضف بدايةً لائحة GunaPanel واجعل خاصية Dock لها نحو اليسار، بعرض 215، ثم أضف لائحة أخرى واجعل خاصية Dock لها نحو الأعلى، بارتفاع 30، ثم أضف أخرى بشكل مشابه ولكن بارتفاع 200.

استعن بالجدول التالي:

القيمة	الخاصية	الأداة
None	AutoScaleMode	Form1
Segoe UI, 12pt	Font	
None	FontBorderStyle	
900, 550	Size	
CenterScreen	Start Position	

21, 101, 192	BackColor	gunaAdvenceButton1
25, 118, 210	BaseColor	
RadioButton	ButtonType	
True	Checked	
46, 144, 241	CheckedBaseColor	
20	ImageOffsetX	
40, 132, 221	LineColor	
1	LineTop	
61, 155, 249	OnHoverBaseColor	
46, 137, 226	OnHoverLineColor	
215, 60	Size	
OVERVIEW	Text	
10	TextOffsetX	
بالمثل لدينا ثلاث أدوات أخرى، مع فارق في خاصية Text وخاصية Checked		
True	Animated	gunaButtonDownload & gunaButtonJoinUs
66, 156, 245	BaseColor	
Segoe UI Light, 12pt	Font	
None	Image	
89, 168, 255	OnHoverBaseColor	
White	OnHoverBorderColor	
White	OnPressedColor	
DOWNLOAD NOW	Text	
Center	TextAlign	
True	Animated	gunaButtonUpdate

	BaseColor	25, 118, 210
	BorderColor	89, 168, 255
	BorderSize	1
	Font	Segoe UI Light, 12pt
	Image	None
	OnHoverBaseColor	89, 168, 255
	OnHoverBorderColor	White
	OnPressedColor	White
	Text	Check for updates!
	TextAlign	Center
gunaControlBox1	BackColor	33, 150, 243
	Dock	Right
	OnHoverBackColor	46, 137, 226
	هناك أداة أخرى من هذا النوع تختلف بالخاصية ControlBoxType	
gunaDragControl1	TargetControl	gunaPanel2
gunaDragControl2	TargetControl	gunaPanel3
gunaLinePanel1	BackColor	Gainsboro
	Dock	Bottom
	LineColor	LightGray
	LineTop	3
gunaPanel1	BackColor	21, 101, 192
	Dock	Left
	Width	215
gunaPanel1	BackColor	33, 150, 243

Top	Dock	
30	Height	
33, 150, 243	BackColor	gunaPanel1
Top	Dock	
200	Height	
White	BackColor	gunaTileButton1
66, 156, 245	BaseColor	
100, 181, 246	OnHoverBaseColor	
Start a course	Text	
10	TextImageOffsetY	
White	BackColor	gunaTileButton2
89, 168, 255	BaseColor	
89, 140, 255	OnHoverBaseColor	
Check your timetable	Text	
10	TextImageOffsetY	
White	BackColor	gunaTileButton3
57, 182, 255	BaseColor	
57, 150, 255	OnHoverBaseColor	
Check your timetable	Text	
10	TextImageOffsetY	

فضلًا عما سبق، استخدم الكود:



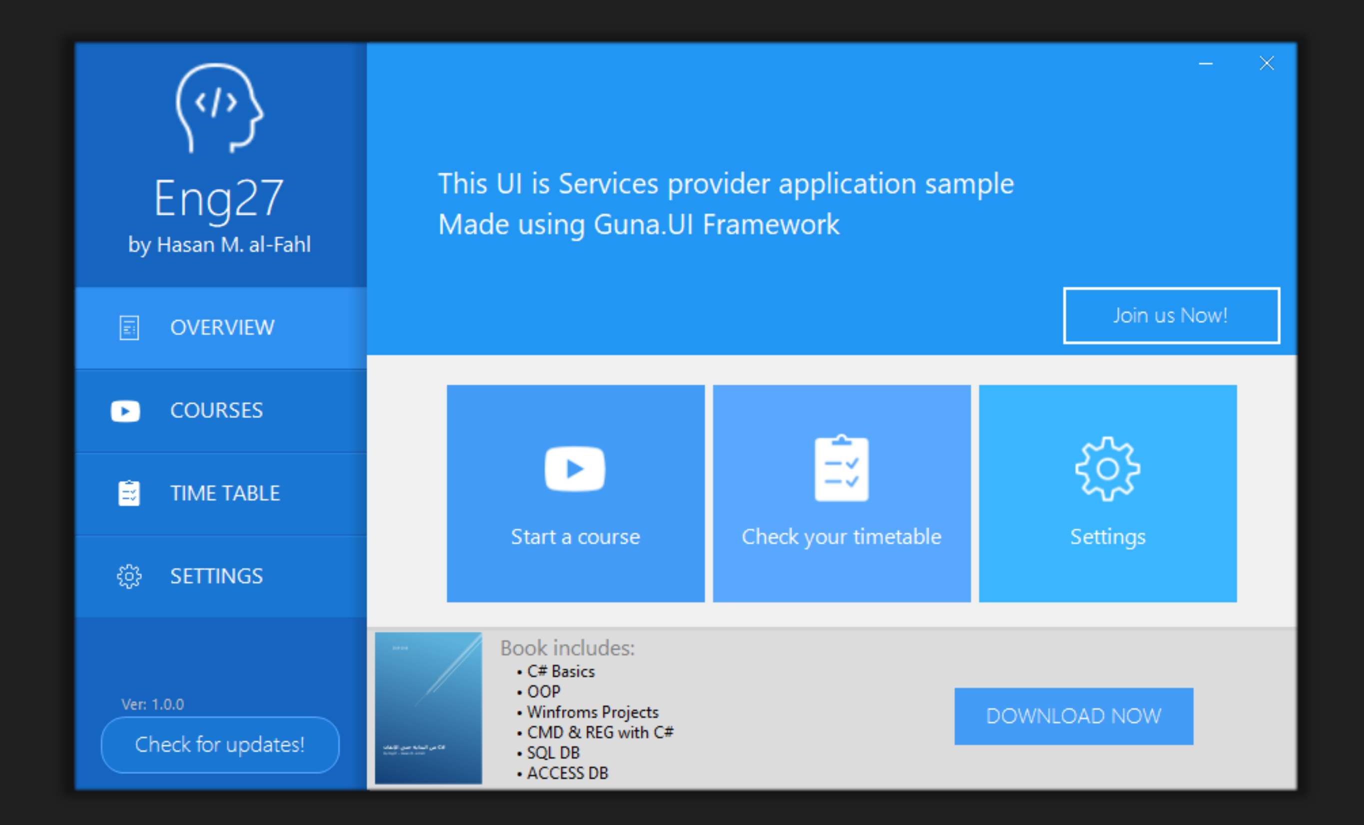
```
uusing System;
using System.Drawing;
using System.Windows.Forms;
using Guna.UI.Lib;

namespace GunaUITest3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            // إظهار ظل على حواف النافذة
            GraphicsHelper.ShadowForm(this);

            // إظهار ظل بين اللائحة1 وأدواتها وبين بقية اللوائح من البرنامج مما يعطي مظهرًا رائعًا!!
            GraphicsHelper.DrawLineShadow(gunaPanel1,
                Color.Black,
                40,
                13,
                Guna.UI.WinForms.VerHorAlign.VerticalRight);

            GraphicsHelper.DrawLineShadow(gunaAdvenceButton1,
                Color.Black,
                40,
                13,
                Guna.UI.WinForms.VerHorAlign.VerticalRight);
            GraphicsHelper.DrawLineShadow(gunaAdvenceButton2,
                Color.Black,
                40,
                13,
                Guna.UI.WinForms.VerHorAlign.VerticalRight);
            GraphicsHelper.DrawLineShadow(gunaAdvenceButton3,
                Color.Black,
                40,
                13,
                Guna.UI.WinForms.VerHorAlign.VerticalRight);
            GraphicsHelper.DrawLineShadow(gunaAdvenceButton4,
                Color.Black,
                40,
                13,
                Guna.UI.WinForms.VerHorAlign.VerticalRight);
        }
    }
}
```

شغل التطبيق لتحصل على:



لاحظ جمالية وأناقة وإبداع وبساطة أدوات منصة Guna.UI! لاحظ تأثير الظل على النافذة، للوهلة الأولى ستشعر أن اللائحة panel1 وأدواتها موجودة في طبقة أسفل من اللوائح الأخرى! لاحظ أيضًا الظل الموجود حول النافذة.. قد يعتقد البعض – وقد حدث فعلًا – أن التطبيق مبني بتقنيات الويب أو WPF، مع أنه مبني باستخدام أدوات WinForms!!

لا داعي لتذكيرك بخاصية الرسوم المتحركة التي تتمتع بها أزرار نافذتك صحيح؟

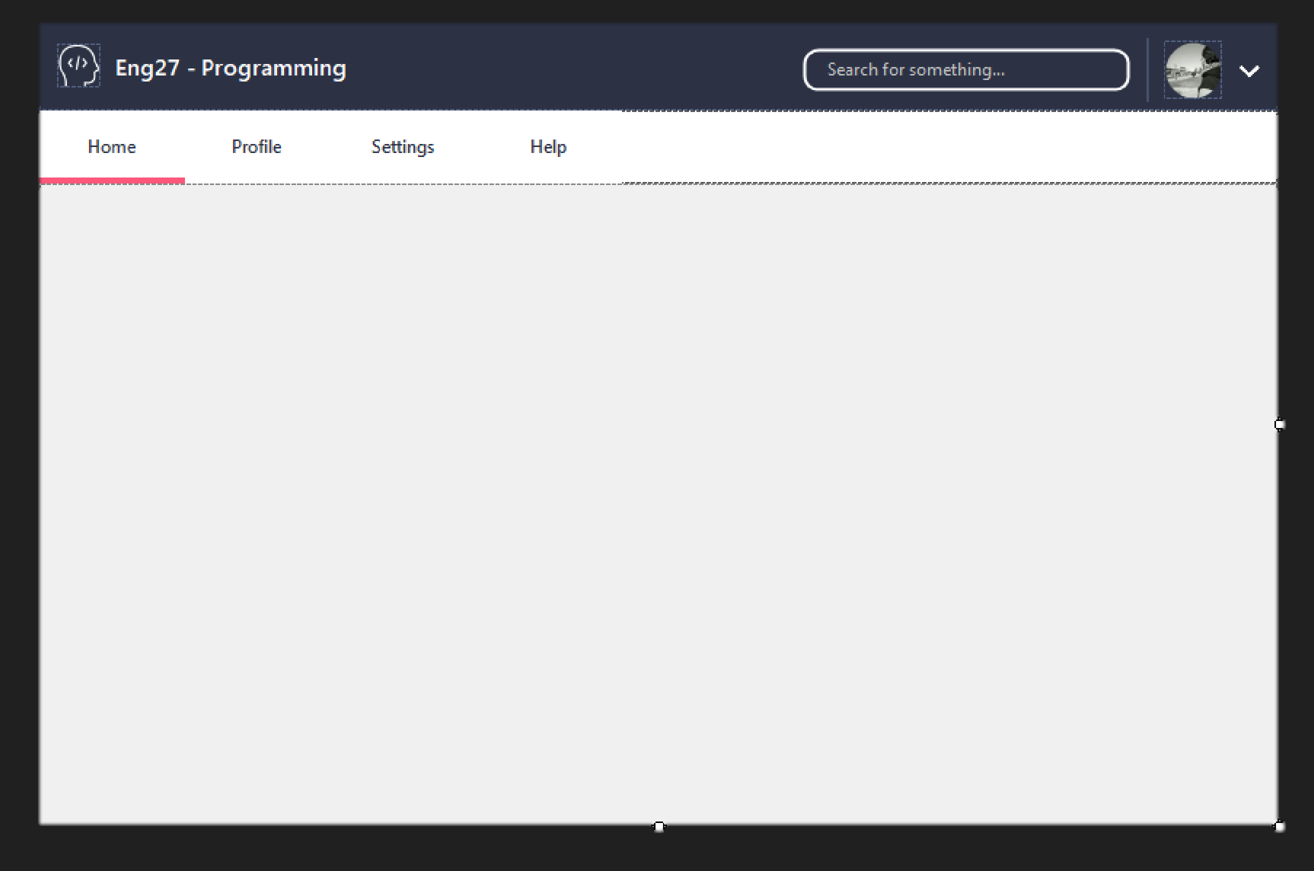
تأمل النافذة وألوانها، والخطوط المستخدمة فيها، وحجوم أدواتها ومواقعها، والأبعاد الفاصلة بينها.. لاحظ أن التطبيق مبني على لونين فقط، الأبيض والأزرق، بالإضافة لمشتقات الأزرق لتمييز الأمور عن بعضها. تذكر دومًا: خير الكلام ما قل ودل، وعليه في التصميم: خير الألوان ما قلت ودلت!!

وفي الختام، قد تتساءل عن سبب وجود panel2 وpanel3 مع أن واحدة منهما تفي بالغرض، لدرجة أنه بعد تشغيل التطبيق لا يمكن التمييز بينهما بسبب لونهما الواحد.

كما أننا خصصنا أداتين DragControl لهما.. وفي الواقع فمعك حق، لا داعي لكليهما، تكفي واحدة منهما، لكنني في البداية أردت تطوير التطبيق وإضافة بعض القوائم عليه ثم عدلت الخطة لأن المشروع كان سيزداد تعقيدًا (وأثناء عودتي للخطة القديمة نسيت حذف إحدى اللائحتين والاكتفاء بواحدة 😅🤦‍♂️).

لوحة تحكم بسيطة

صمم النافذة التالية:



استعن بالجدول التالي لضبط الأدوات:

القيمة	الخاصية	الأداة
None	AutoScaleMode	Form1
Segoe UI, 12pt	Font	
None	FontBorderStyle	
850, 550	Size	

CenterScreen	Start Position	
True	Animated	gunaAdvanedButton1
White	BackColor	
White	BaseColor	
RadioButton	ButtonType	
True	Checked	
White	CheckedBaseColor	
44, 50, 68	CheckedForeColor	
253, 87, 124	CheckedLineColor	
Left	Dock	
Segoe UI Semibold, 9pt	Font	
44, 50, 68	ForeColor	
None	Image	
4	LineButton	
White	OnHoverBaseColor	
44, 50, 68	OnHoverForeColor	
253, 87, 124	OnHoverLineColor	
0	OnPressedDepth	
100, 50	Size	
Home	Text	
هذه الأداة يوجد مثلها ثلاث أدوات مع اختلاف قيمة Checked = False لبقية الأدوات وقيم خاصية Text		
40, 40	Size	gunaCirclePictureBox1
gunaPanel1	TargetControl	gunaDragControl1
Zoom	BackgroundImageLayout	gunaImageButton1

15, 15	Size	
44, 50, 68	BackColor	gunaPanel1
Top	Dock	
60	Height	
White	BackColor	gunaPanel2
Top	Dock	
50	Height	
Control	BackColor	gunaPanel3
Fill	Dock	
Zoom	BackgroundImageLayout	gunaPictureBox1
30, 30	Size	
Zoom	SizeMode	
44, 50, 68	BackColor	gunaTextBox1
White	BorderColor	
2	BorderSize	
44, 50, 68	FocusedForeColor	
White	FocusedBaseColor	
White	FocusedBorderColor	
225, 30	Size	
Search for something...	Text	
10	TextOffsetX	
70, 80, 108	LineColor	gunaVSeparator1
11, 44	Size	
2	Thickness	

استخدم الكود:



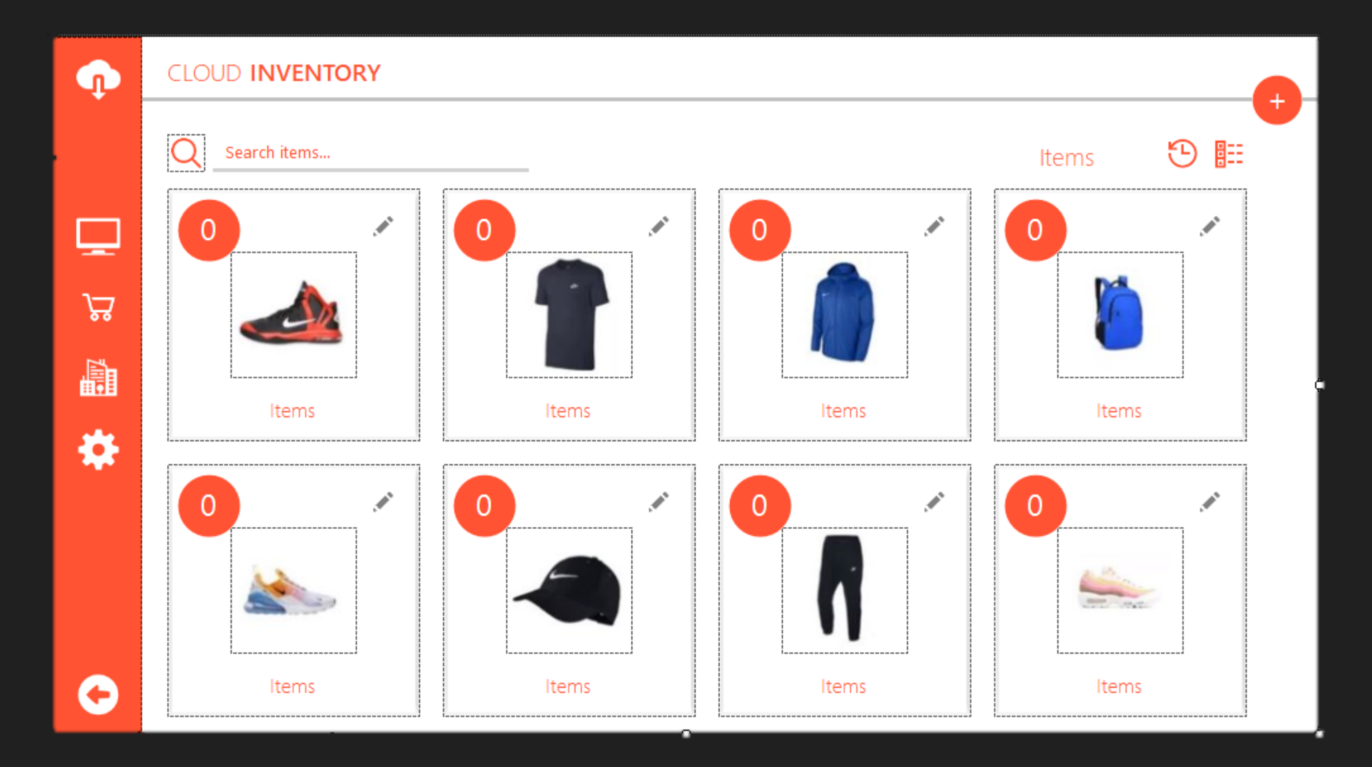
```
using Guna.UI.Lib;
using System;
using System.Drawing;
using System.Windows.Forms;

namespace GunaUITest4
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            GraphicsHelper.ShadowForm(this);
            GraphicsHelper.DrawLineShadow(gunaPanel3,
                Color.Black,
                20,
                10,
                Guna.UI.WinForms.VerHorAlign.HoriziontalTop);
        }
    }
}
```

شغّل البرنامج ولاحظ النتائج:

نافذة بسيطة لإدارة منتجات

صمم النافذة التالية:



استخدم القيم الموضحة بالجدول التالي:

<table>
<tr><th>الأداة</th><th>الخاصية</th><th>القيمة</th></tr>
<tr><td rowspan="5">Form1</td><td>AutoScaleMode</td><td>None</td></tr>
<tr><td>Font</td><td>Segoe UI, 12pt</td></tr>
<tr><td>FontBorderStyle</td><td>None</td></tr>
<tr><td>Size</td><td>1000, 550</td></tr>
<tr><td>Start Position</td><td>CenterScreen</td></tr>
<tr><td rowspan="4">gbEdit1</td><td>BackColor</td><td>Transparant</td></tr>
<tr><td>BackgroundImageLayout</td><td>Zoom</td></tr>
<tr><td>BaseColor</td><td>Transparent</td></tr>
<tr><td>Cursor</td><td>Hand</td></tr>
</table>

	Font	Segoe UI, 9pt
	Location	160, 20
	OnHoverBaseColor	Transparent
	OnPressedDepth	10
	Size	20, 20
	Text	
	هذه الأداة من نوع GunaButton، وعليها رمز تعديل وهي موجودة ضمن gunaShadowPanel يوجد من هذه الأداة 7 أدوات أخرى	
gcbCounter1	Animated	True
	BaseColor	255, 81, 53
	Font	Segoe UI, 18pt
	Location	8, 8
	OnHoverBaseColor	255, 81, 53
	Size	50, 50
	Text	0
	هذه الأداة من نوع GunaCircularButton، وهي زر باللون الأحمر بداخلها عدد بلون أبيض، موجودة ضمن gunaShadowPanel ويوجد منها 7 أدوات أخرى	
glblItem1	AutoSize	False
	Font	Segoe UI Light, 12pt
	ForeColor	255, 81, 53
	Location	8, 164
	Size	185, 25
	Text	Items

<table dir="rtl">
<tr><td rowspan="2"></td><td>TextAlign</td><td>TopCenter</td></tr>
<tr><td colspan="2">هذه الأداة من نوع GunaLabel، موجودة ضمن gunaShadowPanel ويوجد منها 7 أدوات أخرى</td></tr>
<tr><td rowspan="4">gpbPic1</td><td>Location</td><td>50, 50</td></tr>
<tr><td>Size</td><td>100, 100</td></tr>
<tr><td>SizeMode</td><td>Zoom</td></tr>
<tr><td colspan="2">هذه الأداة من نوع GunaPictureBox، موجودة ضمن gunaShadowPanel ويوجد منها 7 أدوات أخرى
تمت إضافة الصور إلى هذه الأداة يدويًّا لإظهار التصميم، وفي الواقع يجب تحميل الصور برمجيًّا</td></tr>
<tr><td rowspan="8">gunaButton1</td><td>Animated</td><td>True</td></tr>
<tr><td>BackColor</td><td>Transparent</td></tr>
<tr><td>BackgroundImageLayout</td><td>Zoom</td></tr>
<tr><td>BaseColor</td><td>Transparent</td></tr>
<tr><td>OnHoverBaseColor</td><td>Transparent</td></tr>
<tr><td>OnPressedDepth</td><td>10</td></tr>
<tr><td>Size</td><td>35, 35</td></tr>
<tr><td colspan="2">يوجد من هذه الأداة 5 أدوات أخرى، وكلها موجودة ضمن اللائحة gunaPanel1 (اللائحة اليسرى)
يوجد أداتان من نفس هذه الأداة موجودتان في الجزء العلوي الأيمن من النافذة، وبنفس الخصائص السابقة إلا أنهما يختلفان بالحجم 30, 30</td></tr>
<tr><td rowspan="3">gunaCircleButton1</td><td>Animated</td><td>True</td></tr>
<tr><td>BaseColor</td><td>255, 81, 53</td></tr>
<tr><td>Font</td><td>Segoe UI, 18pt</td></tr>
</table>

948, 30	Location	
255, 81, 53	OnHoverBaseColor	
40, 40	Size	
+	Text	
Form1	TargetControl	gunaDragControl1
Segoe UI Light, 14pt	Font	gunaLabel1
255, 81, 53	ForeColor	
CLOUD	Text	
Segoe UI Semibold, 14pt, style=Bold	Font	gunaLabel2
255, 81, 53	ForeColor	
INVENTORY	Text	
Segoe UI Light, 14pt	Font	gunaLabel3
255, 81, 53	ForeColor	
Items	Text	
255, 81, 53	FocusedLineColor	gunaLineTextBox1
Segoe UI, 9.75pt	Font	
255, 81, 53	ForeColor	
209, 209, 209	LineColor	
3	LineSize	
250, 30	Size	
Search items...	Text	
10	TextOffsetX	
255, 81, 53	BackColor	gunaPanel1
Left	Dock	

<table>
<tr><td></td><td>Width</td><td>70</td></tr>
<tr><td rowspan="4">gunaPictureBox1</td><td>Size</td><td>30, 30</td></tr>
<tr><td>SizeMode</td><td>Zoom</td></tr>
<tr><td colspan="2">هذه الأداة عليها صورة أداة بحث</td></tr>
<tr><td rowspan="3">gunaSeparator1</td><td>LineColor</td><td>Silver</td></tr>
<tr><td>Size</td><td>942, 10</td></tr>
<tr><td>Thickness</td><td>3</td></tr>
<tr><td rowspan="5">gunaShadowPanel1</td><td>Location</td><td>90, 120</td></tr>
<tr><td>ShadowDepth</td><td>50</td></tr>
<tr><td>ShadowShift</td><td>3</td></tr>
<tr><td>Size</td><td>200, 200</td></tr>
<tr><td colspan="2">يوجد من هذه الأداة 7 أدوات أخرى، وهي لوائح تحتضن أدوات أخرى.. البعد بين اللائحة والأخرى 18 بالاتجاه الطولي والعرضي
أي، يجب أن يكون هناك فراغ بين اللوائح أفقيًّا ورأسيًّا 18 نقطة.. ولاختصار الموضوع عليك اعتمد على المعادلة التالية لمعرفة إحداثيات اللوائح:
$X = 90 + (n - 1) \cdot (200 + 18)$
$Y = 120 + m \cdot (200 + 18)$
حيث n رقم اللائحة المراد إيجاد إحداثياتها، وm تأخذ القيمة 1 للوائح الصف الثاني، وإلا معدومة
فمثلًا اللائحة 3 من الصف 2 إحداثياتها 526, 338</td></tr>
</table>

استخدم الكود التالي:



```
using Guna.UI.Lib;
using Guna.UI.WinForms;
using System.Drawing;
using System.Windows.Forms;

namespace GunaUITest5
{
    public partial class Form1 : Form
    {
        string[] items;
        int[] count;
        int sum = 0;

        public Form1()
        {
            InitializeComponent();
            GraphicsHelper.DrawLineShadow(gunaSeparator1,
                Color.Black,
                20,
                20,
                VerHorAlign.HorizontalBottom);

            // من المفترض الحصول على القيم التالية من قاعدة بيانات مثلًا
            items = new string[] { "Air Max Hyper Agressor",
                "Nike T-Shirt Black",
                "Nike Jacket Blue",
                "School Bag Blue",
                "Air Max Orange and Blue",
                "Nike Cap Black",
                "Nike Long Joger Black",
                "Nike Air Max"};

            count = new int[] { 10, 2, 0, 5, 10, 5, 10, 9 };

            gcbCounter1.Text = count[0].ToString();
            glblItem1.Text = items[0];

            gcbCounter2.Text = count[1].ToString();
            glblItem2.Text = items[1];

            gcbCounter3.Text = count[2].ToString();
            glblItem3.Text = items[2];

            gcbCounter4.Text = count[3].ToString();
            glblItem4.Text = items[3];

            gcbCounter5.Text = count[4].ToString();
            glblItem5.Text = items[4];

            gcbCounter6.Text = count[5].ToString();
            glblItem6.Text = items[5];

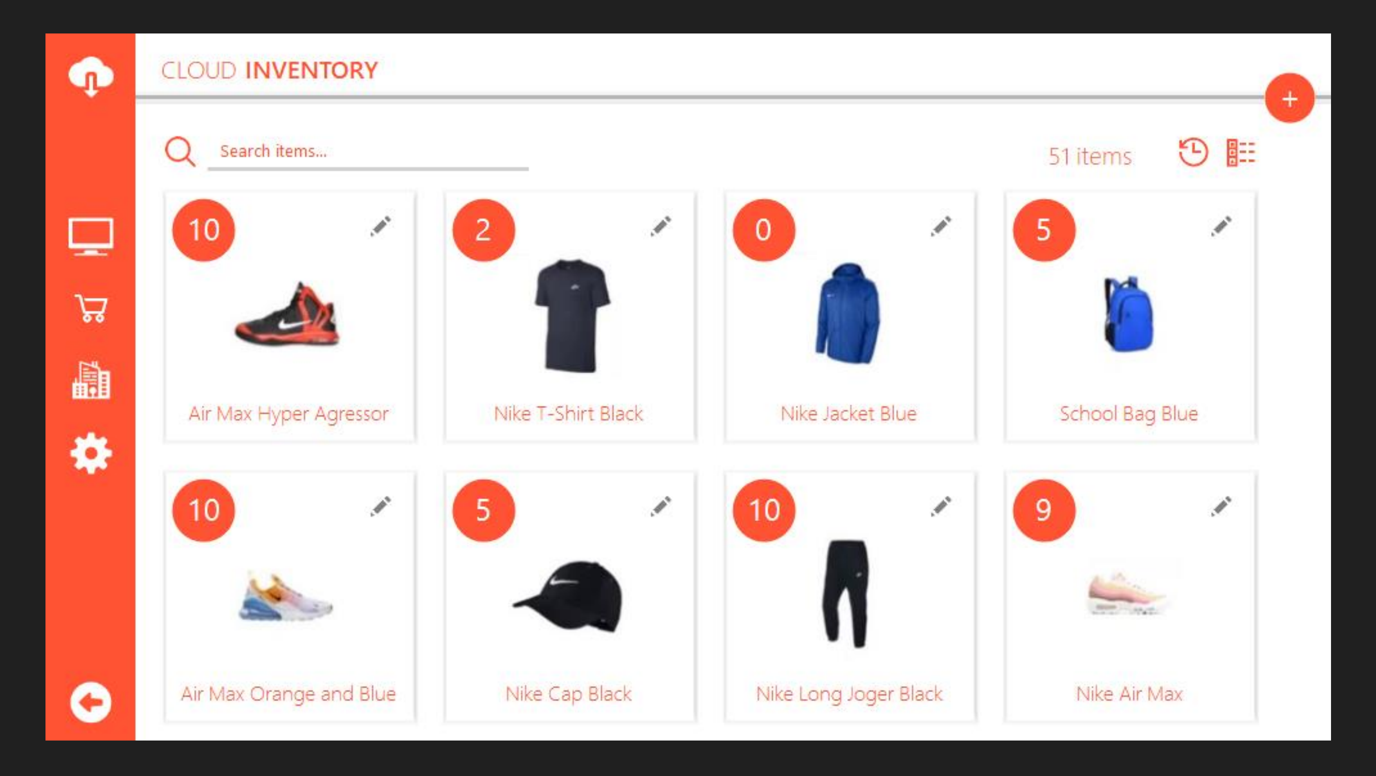
            gcbCounter7.Text = count[6].ToString();
            glblItem7.Text = items[6];

            gcbCounter8.Text = count[7].ToString();
            glblItem8.Text = items[7];
```

```
            foreach (int i in count)
                sum += i;

            gunaLabel3.Text = sum + " items";
        }
    }
}
```

شغّل البرنامج:



نافذة تسجيل دخول / إنشاء حساب

يمكنك من خلال منصة Guna.UI تصميم نافذة بحيث تستخدمها لتسجيل الدخول وتسجيل الخروج في الوقت ذاته، وذلك اعتمادًا على أوامر إظهار وإخفاء الأدوات.

صمم النافذة التالية:

استعن بالجدول التالي:

الأداة	الخاصية	القيمة
	AutoScaleMode	None
	Font	Segoe UI, 12pt
Form1	FontBorderStyle	None
	Size	700, 450
	Start Position	CenterScreen

True	Animated	gunaButton1
30, 231, 185	BaseColor	
30, 231, 185	BorderColor	
2	BorderSize	
Segoe UI, 9.75pt, style=Bold	Font	
White	ForeColor	
None	Image	
White	OnHoverBaseColor	
30, 231, 185	OnHoverBorderColor	
30, 231, 185	OnHoverForeColor	
15	Radius	
100, 35	Size	
SIGN IN	Text	
Center	TextAlign	
True	Animated	gunaButton2
White	BaseColor	
White	BorderColor	
2	BorderSize	
Segoe UI, 9.75pt, style=Bold	Font	
30, 231, 185	ForeColor	
None	Image	
30, 231, 185	OnHoverBaseColor	
White	OnHoverBorderColor	

White	OnHoverForeColor	
15	Radius	
100, 35	Size	
SIGN UP	Text	
Center	TextAlign	
True	Animated	gunaCircleButton1 & gunaCircleButton2 & gunaCircleButton3
White	BaseColor	
1	BorderSize	
Segoe UI, 15.75pt, style=Bold	Font	
Black	ForeColor	
30, 231, 185	OnHoverBaseColor	
50, 50	Size	
f G in	Text	
True	Animated	gunaCircleButton2
30, 231, 185	BaseColor	
White	BorderColor	
1	BorderSize	
Segoe UI, 12pt	Font	
White	ForeColor	
100, 12	Location	
White	OnHoverBaseColor	
30, 231, 185	OnHoverBorderColor	

30, 231, 185	OnHoverForeColor	
30, 30	Size	
x	Text	
gunaPanel2	TargetControl	gunaDragControl1
Segoe UI, 26.25pt, style=Bold	Font	gunaLabel1
30, 231, 185	ForeColor	
Sign in to System	Text	
Segoe UI, 9.75pt	Font	gunaLabel2
Black	ForeColor	
Sign in using social media account	Text	
Segoe UI Light, 26.25pt	Font	gunaLabel3
White	ForeColor	
0, 143	Location	
250, 47	Size	
Hello again!	Text	
MiddleCenter	TextAlign	
Segoe UI, 9.75pt	Font	gunaLabel4
White	ForeColor	
0, 205	Location	
250, 34	Size	
Don't have account? make new one!	Text	

MiddleCenter	TextAlign	
White	BackColor	gunaLineTextBox1 & gunaLineTextBox2
30, 231, 185	FocusedLineColor	
Segoe UI, 12pt	Font	
64, 64, 64	LineColor	
2	LineSize	
375, 32	Size	
Email Password	Text	
30, 231, 185	ActiveLineColor	gunaLinkLabel1
Segoe UI, 9.75pt, style=Bold	Font	
Black	LineColor	
Forgot your password?	Text	
30, 231, 185	BackColor	gunaPanel1
Right	Dock	
250	Width	
White	BackColor	gunaPanel2
Fill	Dock	
30, 30	Size	gunaPictureBox1 & gunaPictureBox2
Zoom	SizeMode	
1000	MaxAnimationTime	gunaTransition1

استخدم الكود:



```
using System;
using System.Windows.Forms;
using Guna.UI.Animation;
using Guna.UI.Lib;
using Guna.UI.WinForms;
using System.Drawing;

namespace GunaUITest6
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            GraphicsHelper.DrawLineShadow(gunaPanel1,
                Color.Black, 50, 20, VerHorAlign.VerticalLeft);

            GraphicsHelper.DrawLineShadow(gunaLabel3,
                Color.Black, 50, 20, VerHorAlign.VerticalLeft);

            GraphicsHelper.DrawLineShadow(gunaLabel4,
                Color.Black, 50, 20, VerHorAlign.VerticalLeft);

            GraphicsHelper.ShadowForm(this);
        }

        private void gunaButton2_Click(object sender, EventArgs e)
        {
            gunaPanel1.Hide();
            if (gunaPanel1.Dock == DockStyle.Right)
            {
                gunaPanel1.Dock = DockStyle.Left;

                GraphicsHelper.DrawLineShadow(gunaPanel1,
                    Color.Black, 50, 20, VerHorAlign.VerticalRight);

                GraphicsHelper.DrawLineShadow(gunaLabel3,
                    Color.Black, 50, 20, VerHorAlign.VerticalRight);

                GraphicsHelper.DrawLineShadow(gunaLabel4,
                    Color.Black, 50, 20, VerHorAlign.VerticalRight);

                gunaLabel1.Text = "Sign up to System";
                gunaLabel2.Text = "Sign up using social media account";
                gunaLabel3.Text = "New here?";
                gunaLabel4.Text = "Have account?" +
                    Environment.NewLine +
                    "sign in!";

                gunaButton1.Text = "SIGN UP";
                gunaButton2.Text = "SIGN IN";
                gunaLinkLabel1.Visible = false;

                gunaTransition1.ShowSync(gunaPanel1, false, Animation.HorizSlide);
            }
```

```
            else if (gunaPanel1.Dock == DockStyle.Left)
            {
                gunaPanel1.Dock = DockStyle.Right;

                GraphicsHelper.DrawLineShadow(gunaPanel1,
                    Color.Black, 50, 20, VerHorAlign.VerticalLeft);

                GraphicsHelper.DrawLineShadow(gunaLabel3,
                    Color.Black, 50, 20, VerHorAlign.VerticalLeft);

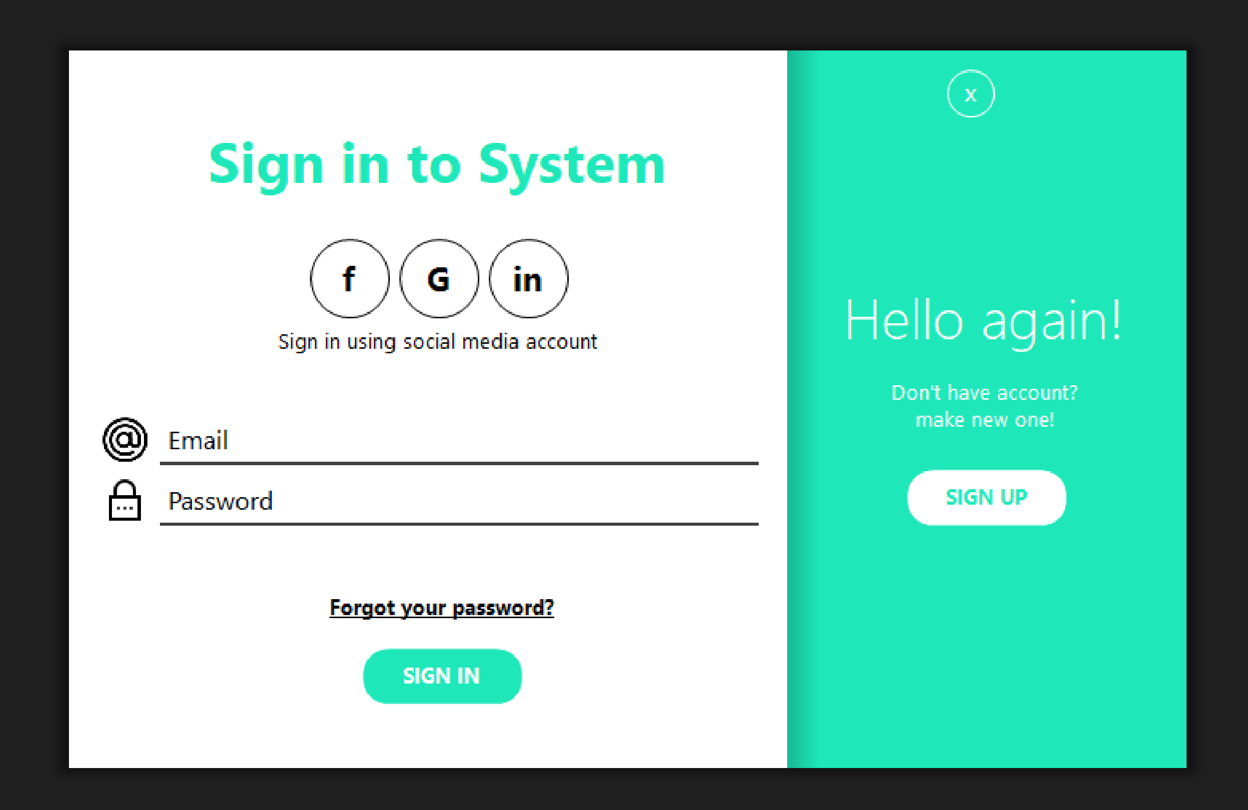
                GraphicsHelper.DrawLineShadow(gunaLabel4,
                    Color.Black, 50, 20, VerHorAlign.VerticalLeft);

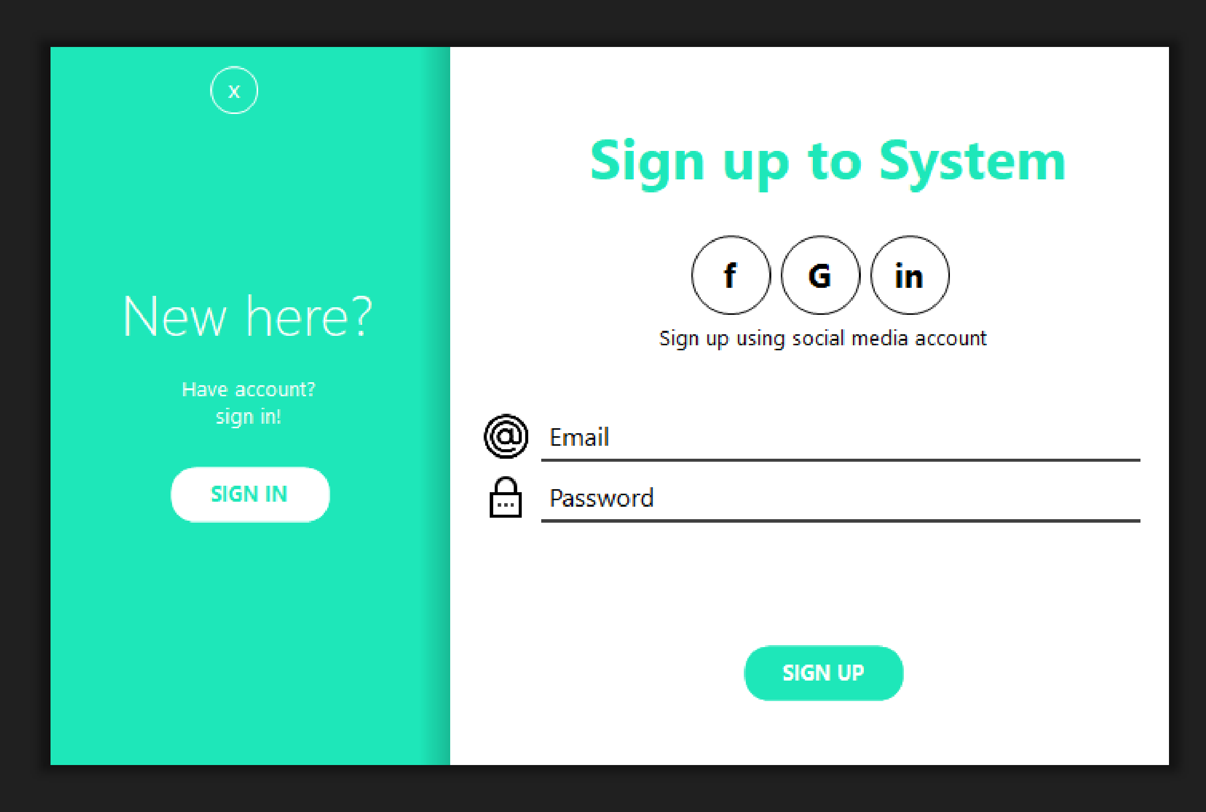
                gunaLabel1.Text = "Sign in to System";
                gunaLabel2.Text = "Sign in using social media account";
                gunaLabel3.Text = "Hello again!";
                gunaLabel4.Text = "Don't have account?" +
                    Environment.NewLine +
                    "make new one!";

                gunaButton1.Text = "SIGN IN";
                gunaButton2.Text = "SIGN UP";
                gunaLinkLabel1.Visible = true;

                gunaTransition1.ShowSync(gunaPanel1, false, Animation.HorizSlide);
            }
        }
    }
}
```

شغل البرنامج، وانقر على زر إنشاء حساب:

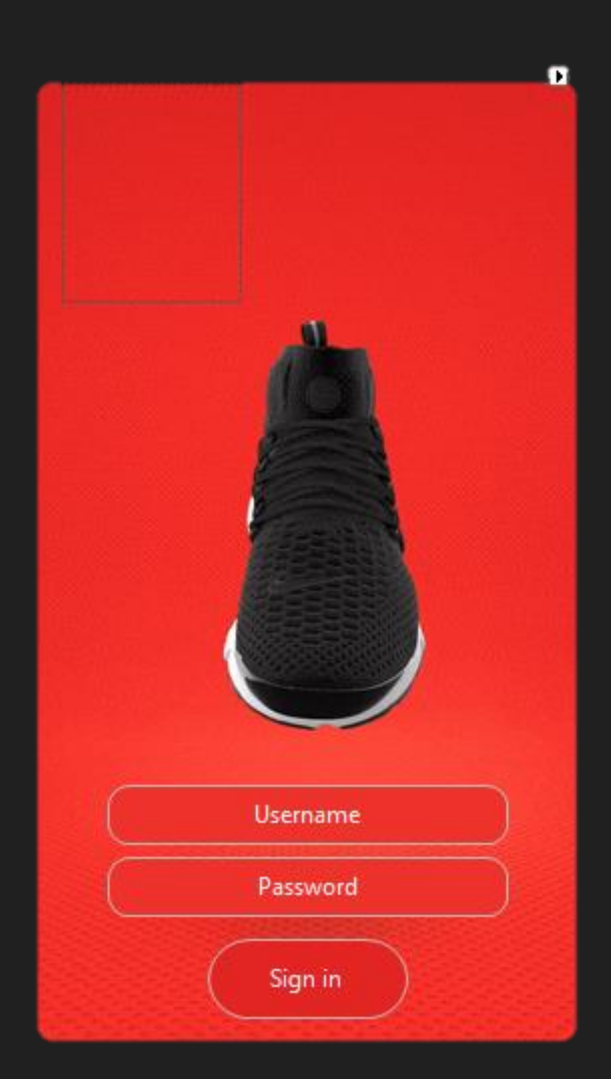




لاحظ انعكاس ألوان الأدوات عند مرور مؤشر الفأرة عليها. لاحظ أن الظل أعطى اللائحة الثانوية gunaPanel1 ذات اللون الغامق تأثيرًا وكأنها خلف (أو تحت) اللائحة الرئيسية gunaPanel2 ذات اللون الفاتح. يمكنك تجربة عكس العملية، بوضع الظل على اللائحة الرئيسية لتجعلها تبدو أدنى من اللائحة الثانية.

نافذة تسجيل دخول متحركة

يمكنك ضبط خلفية النافذة على صورة متحركة GIF، وذلك باستخدام صندوق الصورة GunaPictureBox:



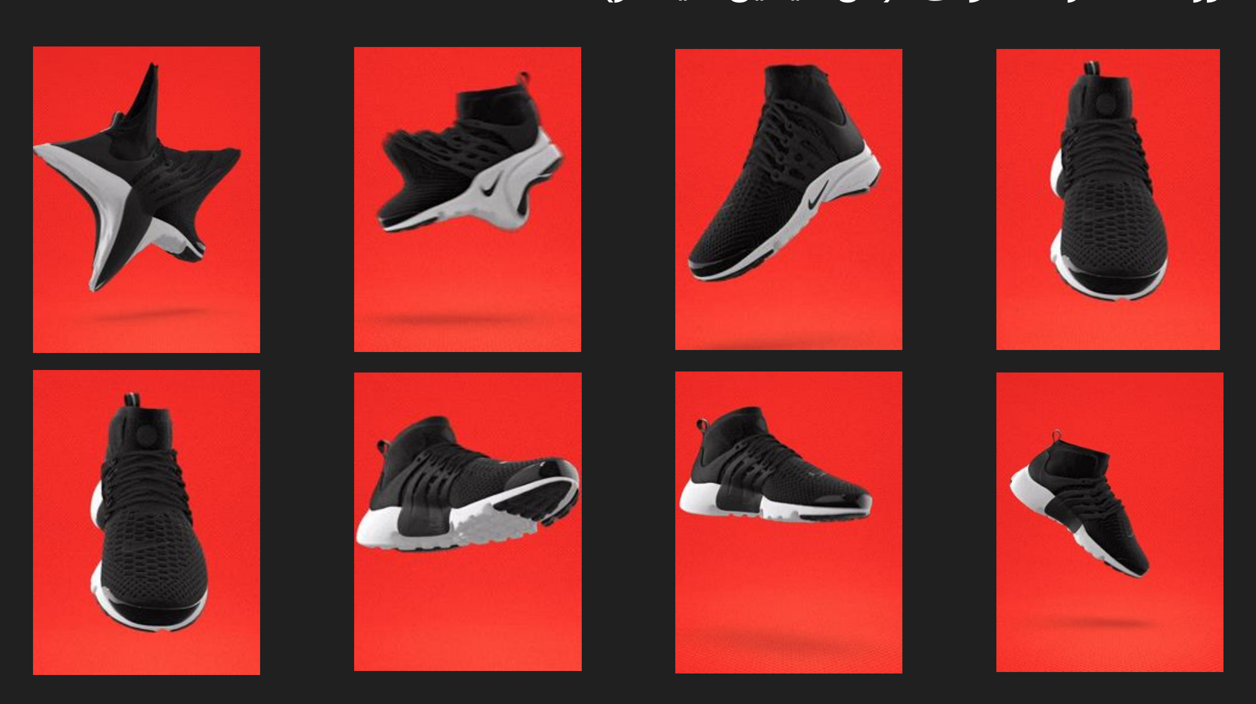
اعتمد على الجدول التالي لضبط الأدوات:

الأداة	الخاصية	القيمة
Form1	AutoScaleMode	None
	BackgroundImage	صورة متحركة 1
	Font	Century Gothic, 8.25pt
	FontBorderStyle	None
	Opacity	90%
	Size	270, 480
	Start Position	CenterScreen
gunaButton1	Animated	True
	BackColor	Transportent
	BaseColor	226, 37, 35
	BorderColor	Silver
	BorderSize	1
	Font	Segoe UI, 9pt
	ForeColor	White
	Image	None
	OnHoverBaseColor	150, 24, 23
	OnHoverBorderColor	Black
	OnHoverForeColor	White
	Radius	20
	Size	100, 40
	Text	Sign in
	TextAlign	Center
gunaTextBox1	BackColor	Transportent

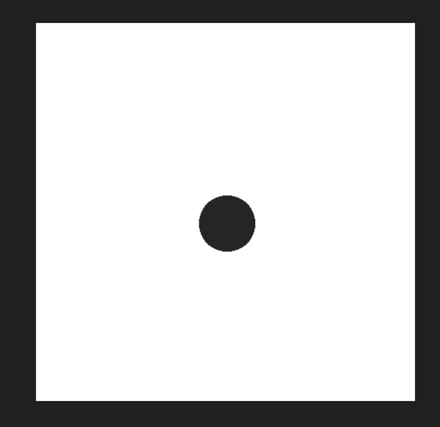
239, 49, 44	BaseColor	
Silver	BorderColor	
1	BorderSize	
239, 49, 44	FocusedBaseColor	
Black	FocusedBorderColor	
White	FocusedForeColor	
Segoe UI, 9pt	Font	
White	ForeColor	
10	Radius	
200, 30	Size	
Username	Text	
Center	TextAlign	
Transportent	BackColor	gunaTextBox2
239, 49, 44	BaseColor	
Silver	BorderColor	
1	BorderSize	
239, 49, 44	FocusedBaseColor	
Black	FocusedBorderColor	
White	FocusedForeColor	
Segoe UI, 9pt	Font	
White	ForeColor	
10	Radius	
200, 30	Size	
Password	Text	
Center	TextAlign	

Fill	Dock	
صورة متحركة 1	Image	gunaPictureBox1
AutoSize	SizeMode	
Transparent	BackColor	gunaPictureBox2
صورة متحركة 2	Image	
12, 0	Location	
90, 110	Size	
Zoom	SizeMode	
5	Radius	gunaEllipse1
gunaPictureBox1	TargetControl	gunaDragControl1

الصورة المتحركة الأولى: (من اليمين لليسار)



الصورة المتحركة الثانية: (من اليمين لليسار)







ولك أن تتخيل كيف سيتحرك ملفا الـ GIF.

وضعنا الصورة المتحركة 1 كخلفية للنموذج Form1 وصندوق الصورة gunaPictureBox1 حتى تصبح خلفية الصورة المتحركة 2 بلون النموذج، فلو لم نقم بهذه الحركة لأصبح اللون الشفاف في صندوق الصورة gunaPictureBox2 بلون الخاصية BackColor للنموذج.

استخدم الكود:



```
using System;
using System.Windows.Forms;
using Guna.UI.Animation;
using Guna.UI.Lib;
using Guna.UI.WinForms;
using System.Drawing;

namespace GunaUITest7
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            GraphicsHelper.ShadowForm(this);
        }
    }
}
```

شغل المشروع ولاحظ:

منصة Bunifu

كلمة Bunifu تعني الإبداع بلغة من اللغات، وهذه المنصة اسم على مسمى. هي منصة مدفوعة، وإصداراتها كثيرة لإصلاح أخطاء الإصدارات السابقة وإضافة مكونات جديدة.



قد تحدث بعض الأخطاء عند استخدام أدوات من إصدارين مختلفين، لذلك يجب الاكتفاء بأدوات إصدار معين دون استخدام أدوات الإصدارات الأخرى.

يمكنك الحصول على دعم من خلال موقع المنصة [1]، كما يمكنك معرفة كيفية استخدام كل أداة بأداتها من خلال الموقع نفسه، ويمكنك التحقق من الجديد من الإصدارات من هنا [2].

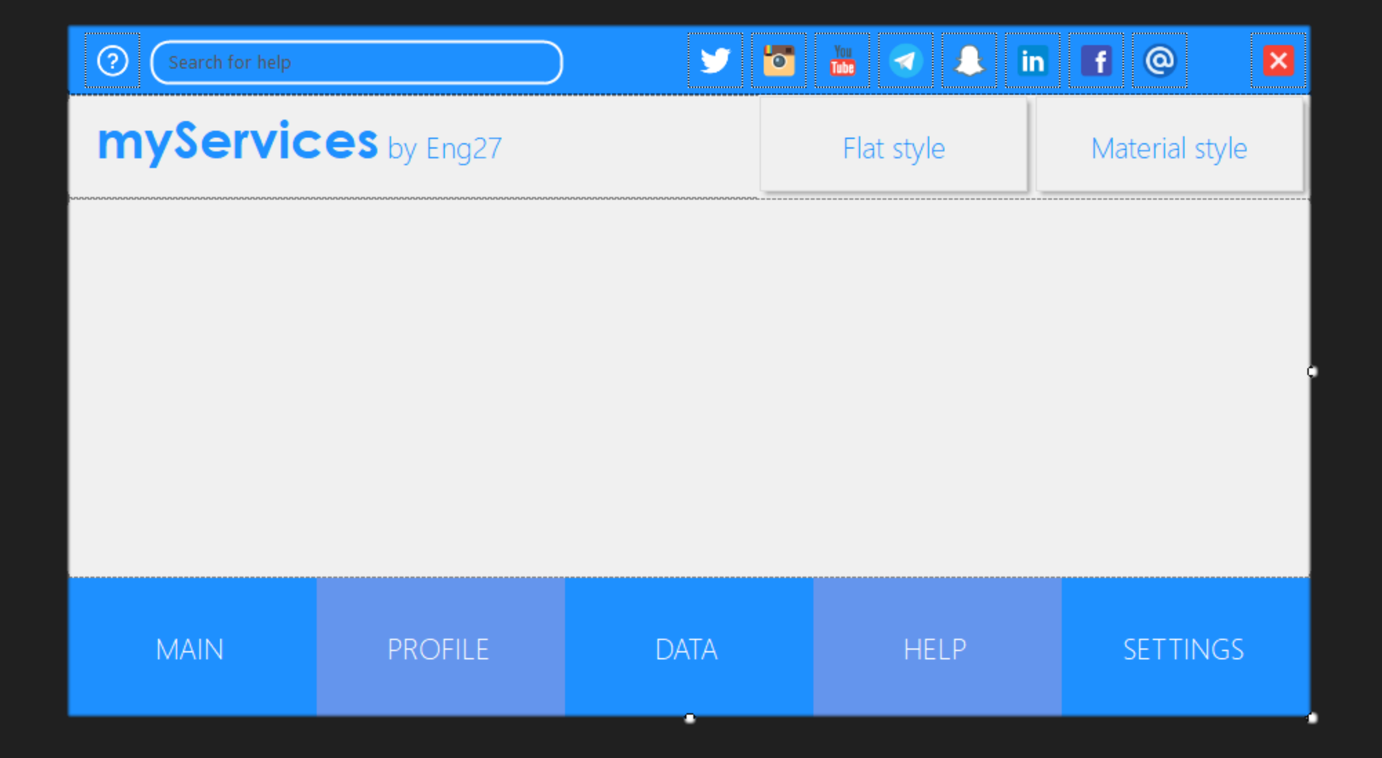
نافذة عرض خدمات

يمكنك إنشاء برنامج صغير لعرض خدماتك أو خدمات شركتك – أو يمكنك استخدامه لأغراض أخرى – وذلك عبر استخدام بضعة أدوات من أدوات Bunifu، كما يمكنك الإضافة على المثال أو التعديل عليه للحصول على تصاميم أخرى. الأدوات المستخدمة هي من الإصدار 1.10.

يوضح لك هذا المثال كيفية استخدام أداة زر الصورة وأداة اللائحة المظللة وأداة التلميح وأداة الانتقالات وأداة تحجيم النافذة FormDock. أداة زر الصورة تستخدم عند عدم الرغبة بوضع نصوص على الأزرار وإنما صور فقط، حيث تكبر الصورة بنسبة معينة عند وقوف مؤشر الفأرة عليها، تمامًا كالأداة ButtonImage التي ألّفناها في الفصل السابع. وتعطيك أداة الانتقالات مجالًا واسعًا من الانتقالات التي يمكن استخدامها لعرض وإخفاء الأدوات.

[1] https://docs.bunifuframework.com/en/.
[2] https://bunifuframework.com/change-log/.

صمم النافذة التالية:



في البداية أنشئ أربعة لوائح Panel، اجعل خاصية Dock للأولى والثانية على القيمة Top، وللثالثة Bottom، وللرابعة Fill.

الجدول التالي فيه الأدوات المستخدمة:

الأداة	الخاصية	القيمة
Form1	AutoScaleMode	None
	Font	Segoe UI, 9.75pt
	FontBorderStyle	None
	Size	900, 500
	Start Position	CenterScreen
bunifuFormDock1	AllowFormDragging	True
	AllowFormDropShadow	True

الأداة	الخاصية	القيمة
	AllowFormResizing	False [1]
	AllowObacity... [2]	False [3]
bunifuImageButton1	Anchor	Top, Right [4]
	ImageMargin	10
	Size	40, 40
	هناك 9 أدوات أخرى من هذا النوع، ولجميعها الخصائص السابقة إلا أداة واحدة (التي لها صورة "مساعدة")	
bunifuShadowPanel1 & bunifuShadowPanel2	Dock	Right
	Width	200
	هاتان الأداتان موجودتان في اللائحة panel2، وموجود ضمن كل واحدة منهما أداة Button	
bunifuTextBox1	BorderColorActive	RoyalBlue
	BorderColorDisabled	Gray
	BorderColorHover	Cyan
	BorderColorIdle	White
	BorderRadius	25
	FillColor	DodgerBlue
	ForeColor	White
	Size	300, 35
	TextPlaceholder	Search for help
bunifuToolTip1	BackColor	32, 32, 32
	BorderColor	64, 64, 64

[1] يمكنك ضبط خاصة السماح بتغيير حجم النافذة بعد استخدام بعض الأكواد لضبط مواقع وحجوم الأدوات الأخرى التي ستتأثر بتغيير وضعية النافذة WindowState، وبعد إضافة أزرار تكبير وتصغير النافذة.
[2] اسم الخاصية هو AllowObacityChangesWhileDragging.
[3] يمكنك ضبطها على True لا مشكلة.
[4] إذا سمحت بتغيير حجم النافذة فيجب ضبط هذه الخاصية على Top, Right، وإلا فوجودها مثل عدمها.

<table>
<tr><td rowspan="3"></td><td>ShowBorders</td><td>True</td></tr>
<tr><td>TextFont</td><td>Segoe UI, 9pt</td></tr>
<tr><td>TitleFont</td><td>Segoe UI Light, 12pt</td></tr>
<tr><td>bunifuTransition1</td><td>AnimationType</td><td>HorizSlide</td></tr>
<tr><td rowspan="2">bunifuTransition2</td><td>AnimationType</td><td>Transparent</td></tr>
<tr><td>MaxAnimation</td><td>1000</td></tr>
<tr><td rowspan="6">button1
&
button2</td><td>Dock</td><td>Fill</td></tr>
<tr><td>Font</td><td>Segoe UI Light,
15.75pt</td></tr>
<tr><td>ForeColor</td><td>DodgerBlue</td></tr>
<tr><td>Text</td><td>Material style
&
Flat style</td></tr>
<tr><td colspan="2">هاتان الأداتان موجودتان في اللائحة الوسطى ضمن أدوات اللائحة المظللة</td></tr>
<tr><td rowspan="8">button3</td><td>BorderColor</td><td>DodgerBlue</td></tr>
<tr><td>Dock</td><td>Left</td></tr>
<tr><td>FlatAppearance.BorderSize</td><td>0</td></tr>
<tr><td>FlatStyle</td><td>Flat</td></tr>
<tr><td>Font</td><td>Segoe UI Light,
15.75pt</td></tr>
<tr><td>ForeColor</td><td>White</td></tr>
<tr><td>Width</td><td>180</td></tr>
<tr><td>Text</td><td>MAIN</td></tr>
</table>

	هناك 4 أدوات أخرى بنفس خصائص هذه الأداة إلا الخاصية Text	
label1	Font	Century Gothic, 27.75pt, style=Bold
	ForeColor	DodgerBlue
	Text	myServices
label2	Font	Segoe UI Light, 15.75pt
	ForeColor	DodgerBlue
	Text	by Eng27
panel1	Dock	Top
	Height	50
panel2	Dock	Top
	Height	75
panel3	Dock	Bottom
	Height	100
panel4	Dock	Fill

وبالنسبة لأقسام البرنامج التي سيتم عرضها عند الضغط على أزرار اللائحة السفلى فصمم 5 أدوات UserControl بما يناسبك من أدوات ومحتوى. ثم استخدم الكود التالي لضبط الانتقال بين أقسام البرنامج:

```
using System;
using System.Windows.Forms;

namespace BunifuTest2
{
    public partial class Form1 : Form
    {
        // استنساخ كائنات تمثل أقسام البرنامج
        Control currentControl;
        UserControl1 uc1 = new UserControl1();
```

```
        UserControl3 uc3 = new UserControl3();
        UserControl2 uc2 = new UserControl2();
        UserControl4 uc4 = new UserControl4();
        UserControl5 uc5 = new UserControl5();

        public Form1()
        {
            InitializeComponent();

            // إضافة أقسام البرنامج إلى اللائحة الوسطى
            uc1.Dock = DockStyle.Fill;
            panel4.Controls.Add(uc1);
            uc2.Dock = DockStyle.Fill;
            panel4.Controls.Add(uc2);
            uc3.Dock = DockStyle.Fill;
            panel4.Controls.Add(uc3);
            uc4.Dock = DockStyle.Fill;
            panel4.Controls.Add(uc4);
            uc5.Dock = DockStyle.Fill;
            panel4.Controls.Add(uc5);
            currentControl = uc1;

            // إخفاء جميع الأقسام
            uc1.Visible = false;
            uc2.Visible = false;
            uc3.Visible = false;
            uc4.Visible = false;
            uc5.Visible = false;
        }

        // الإجراءات الخمسة التالية تقوم بالبداية بإخفاء القسم الحالي وعرض قسم معين، وضبط الأخير على أنه القسم الحالي
        private void button3_Click(object sender, EventArgs e)
        {
            bunifuTransition2.HideSync(currentControl);
            bunifuTransition1.ShowSync(uc1);
            currentControl = uc1;
        }

        private void button4_Click(object sender, EventArgs e)
        {
            bunifuTransition2.HideSync(currentControl);
            bunifuTransition1.ShowSync(uc2);
            currentControl = uc2;
        }

        private void button5_Click(object sender, EventArgs e)
        {
            bunifuTransition2.HideSync(currentControl);
            bunifuTransition1.ShowSync(uc3);
            currentControl = uc3;
        }

        private void button6_Click(object sender, EventArgs e)
        {

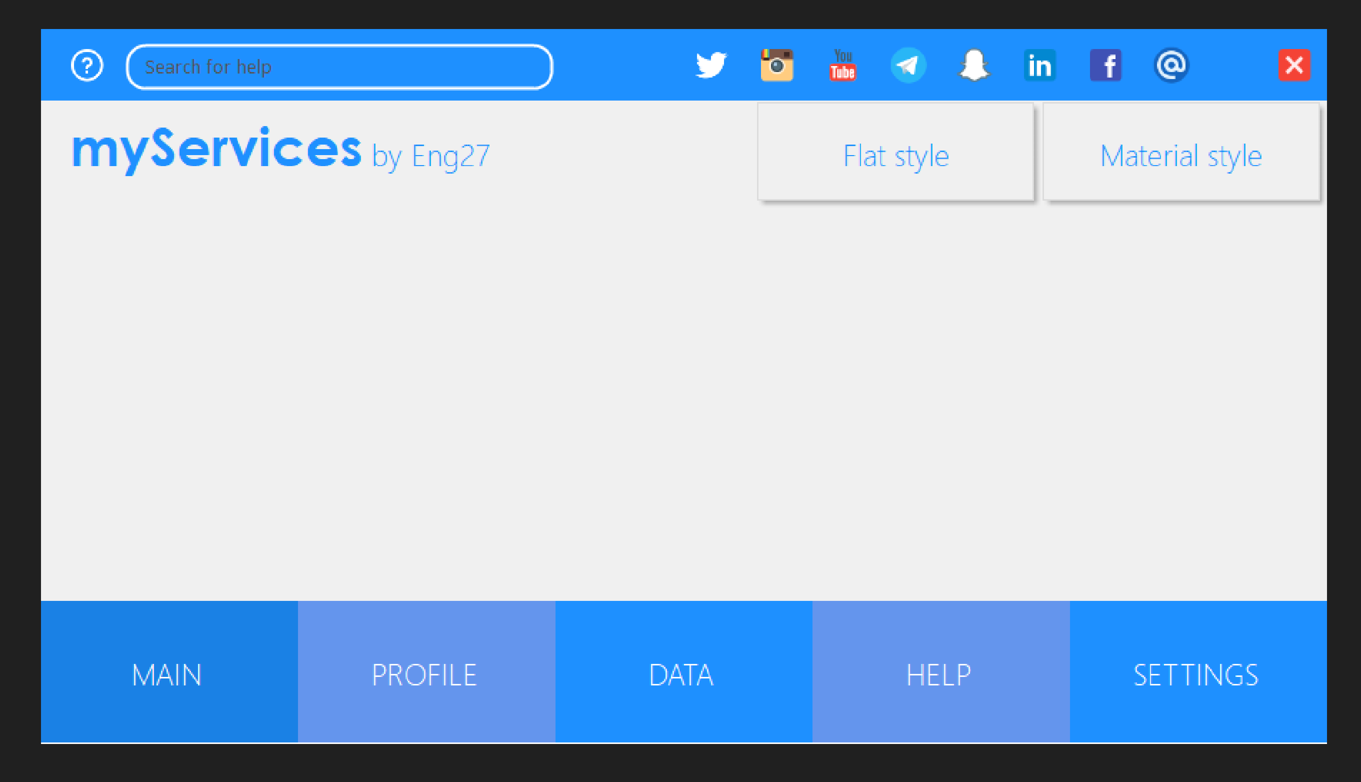
            bunifuTransition2.HideSync(currentControl);
            bunifuTransition1.ShowSync(uc4);
            currentControl = uc4;
        }
```

```
        private void button7_Click(object sender, EventArgs e)
        {

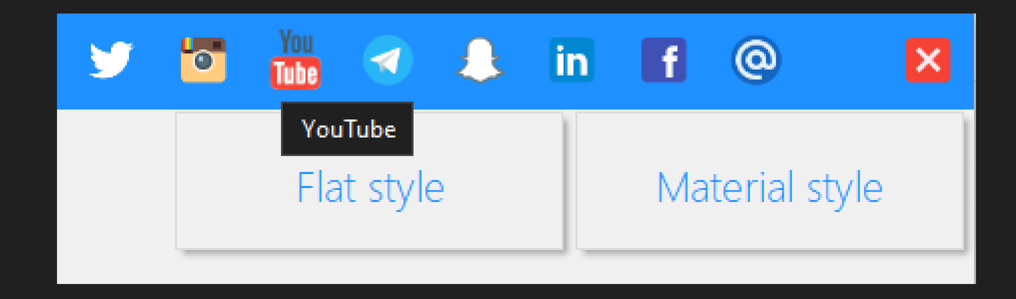
            bunifuTransition2.HideSync(currentControl);
            bunifuTransition1.ShowSync(uc5);
            currentControl = uc5;
        }

    }
}
```

بعد تشغيل البرنامج ستحصل على:

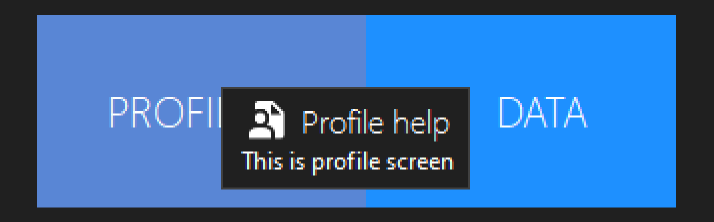


عند وقوف مؤشر الفأرة على أحد الأزرار:

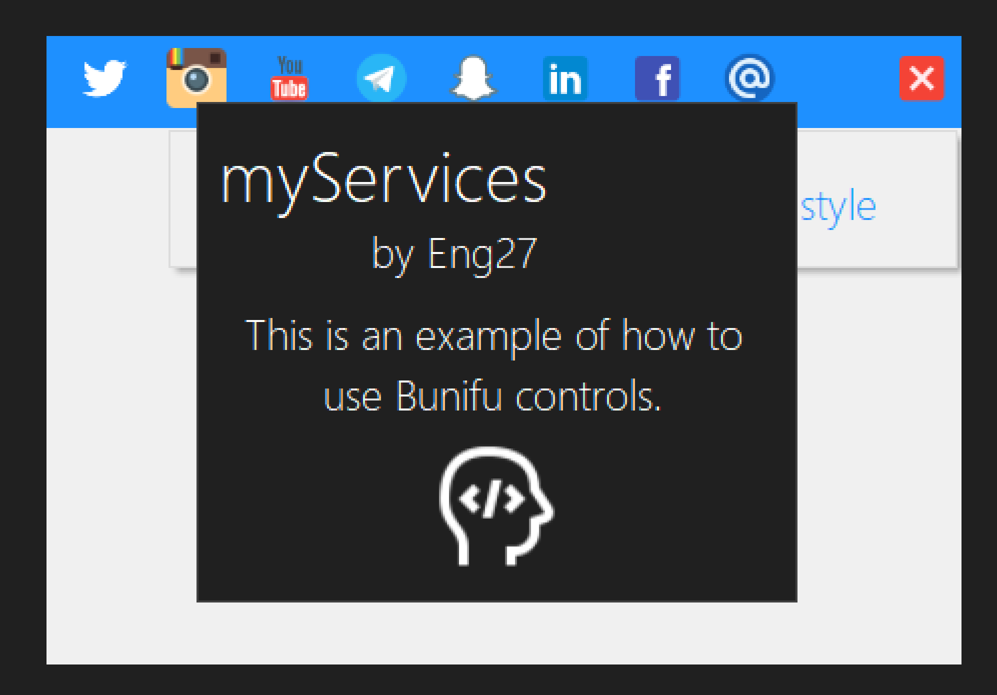


لاحظ كيف يكبر حجم زر الصورة عند الوقوف عليه، ولاحظ الرسالة المنبثقة (التلميح).

لاحظ رسالة التلميح عند إضافة عنوان لها وصورة:



يمكنك تصميم تلميح خاص بك من خلال تصميم أداة مستخدم UserControl وإسنادها إلى الخاصية DisplayControl للأداة BunifuToolTip، لاحظ:



ولهذا صمم الأداة التالية:



الأدوات المستخدمة هي 3 أدوات Label وأداة PictureBox لها الخاصية BackgroundImageLayout مساوية لـ Zoom.

ولهذا يجب إسناد كائن من نوع هذه الأداة إلى الخاصية DisplayControl لأداة التلميح، وذلك في التابع البناء للنافذة Form1.

```
public Form1()
{
    InitializeComponent();

    //.
    //.
    //Codes

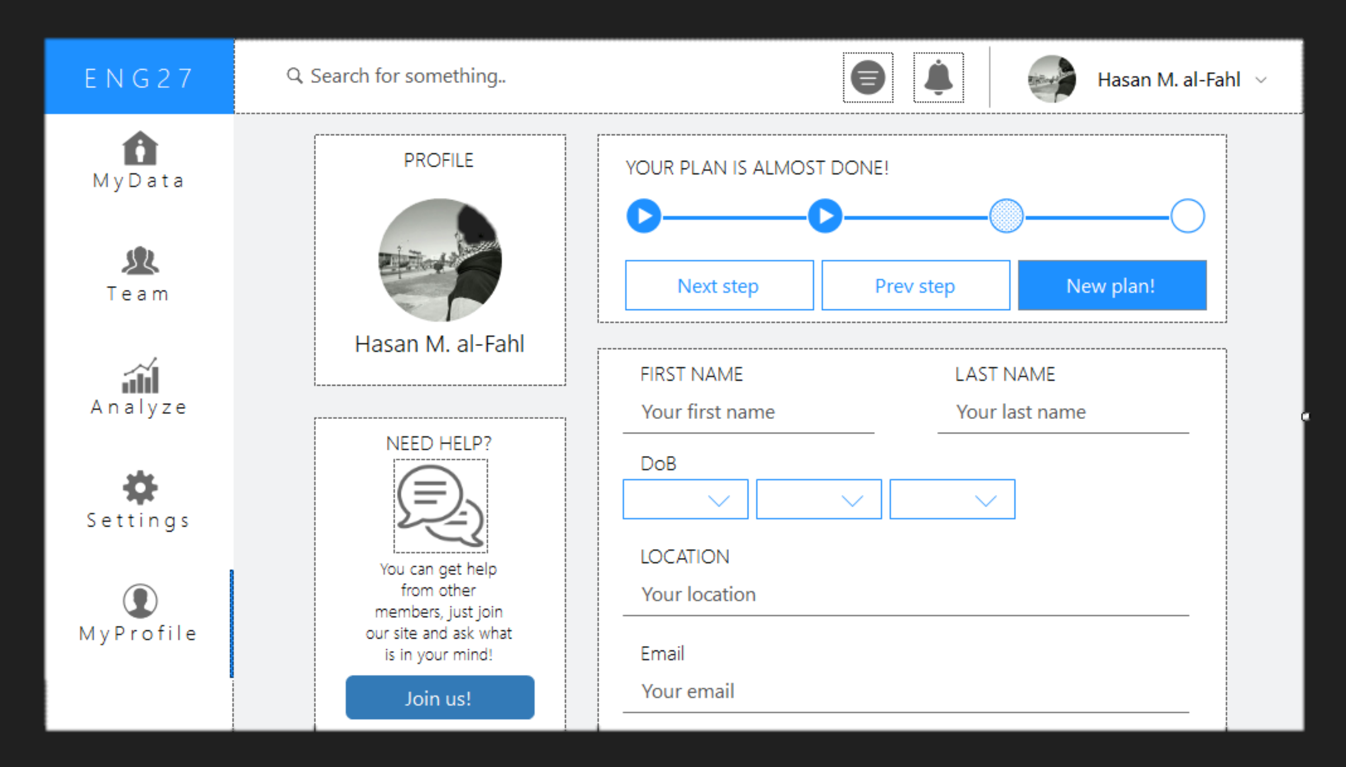
    bunifuToolTip1.DisplayControl = new ToolTipUserControl();
}
```

يمكنك تمرير التفاصيل التي تود عرضها كوسطاء للتابع البناء للـ UserControl التي ستعرضها كتلميح للأدوات.

نافذة ملف شخصي

إن ما يحتاجه المستخدم في ملفه الشخصي هو معرفة بياناته الأساسية وإمكانية تعديلها، وإحصائيات يجمعها البرنامج عن تجربة المستخدم UX.. في هذا المثال لدينا واجهة لتطبيق فيه عدة أقسام، سنتناول قسم الملف الشخصي فقط، وفي الملف الشخصي سنعرض بعض البيانات بأقسام مختلفة (كل قسم بلائحة Panel لتمييز الأقسام عن بعضها)، يمكنك تعديل هذه الأقسام والإضافة عليها وفق حاجتك.. الأدوات المستخدمة من النسخة 1.10.

صمم النافذة التالية:



في البداية أضف لائحة Panel واجعل خاصية Dock لها نحو اليسار، ثم ضع أخرى واجعل خاصية Dock لها نحو الأعلى، ثم أخرى نحو الأعلى داخل الأولى.. وبقية اللوائح موزعة بأماكن معينة سيتم إيضاحها في جدول لاحق.

النافذة تمثل لوحة تحكم فيها – كما تلاحظ – ثلاث أقسام: قسم علوي فيه المستخدم الحالي (مع إمكانية تغييره) والإشعارات وخيارات أخرى مع قسم للبحث ضمن التطبيق، القسم الجانبي الأيسر فيه أزرار التنقل بين أجزاء البرنامج المختلفة، القسم الأوسط فيه الجزء الذي سيختاره البرنامج من القسم الجانبي.

لمثالنا، وكما ذكرنا سابقًا، فإننا سنصمم فقط نافذة الملف الشخصي أي ما سيظهر في القسم الأوسط عند اختيار زر "MyProfile"، وفيه اخترنا بعض الأمور لعرضها، وهي معلومات عن المستخدم الحالي وإمكانية المساعدة والتقدم الحالي وبعض المعلومات الشخصية، وكل ما سبق ليس أكثر من مثال والغاية منه عرض الفكرة، لذلك يمكنك إضافة معلومات أخرى تخدم برنامجك، أو حتى إضافة أقسام أخرى (لاحظ أن النافذة مقسومة إلى أربع لوائح Panels). لاحظ أن البرنامج يقتصر على ألوان معينة متباينة،

فأجزاؤه تأخذ اللون الأبيض كلون للخلفية والأسود كلون للخط، أو الأزرق DodgerBlue كلون للخلفية والأبيض كلون للخط، ولون رمادي قريب للأبيض كخلفية للنافذة. وهذا – أعني اختيار ألوان معينة مدروسة –ما تحدثنا عنه في فصل سابق. لاحظ أيضًا وجود خط أزرق اللون في القسم الجانبي من البرنامج للدلالة على الجزء المختار من البرنامج، وبذلك فقد حققنا السهولة (UX) والجمالية (UI) بخطوة بسيطة جدًا!!

استعن بالجدول التالي لضبط خصائص أدوات البرنامج:

الأداة	الخاصية	القيمة
Form1	AutoScaleMode	None
	BackColor	241, 242, 243
	Font	Segoe UI, 12pt
	FontBorderStyle	None
	Size	1000, 600
	Start Position	CenterScreen
bunifuButton1	ButtonText	Join us!
	ForeColor	While
	IdleBorderColor	51, 122, 183
	IdleBorderRaduis	10
	IdleFileColor	51, 122, 183
bunifuDropdown1	Color	DodgerBlue
	ForeColor	DodgerBlue
	IndicatorColor	DodgerBlue
	ItemForeColor	DodgerBlue
	من المفترض إضافة أيام الشهر لعناصر هذه الأداة هناك أداتين بنفس الخصائص السابقة، إلا أنهما للأشهر والسنوات	

الأداة	الخاصية	القيمة
bunifuFormDock1	AllowFormDragging	True
	AllowFormDropShadow	True
	AllowFormResizing	False [1]
	AllowObacity... [2]	False [3]
bunifuImageButton1	ImageMargin	10
	Size	40, 40
	هناك أداة ثانية بالخصائص نفسها، وتختلف بالصورة.	
bunifuPictureBox1	BorderRaduis	20
	Location	630, 12
	Size	40, 40
	هذه الأداة موجودة في panel2	
bunifuPictureBox2	BorderRaduis	50
	Location	50, 50
	Size	100, 100
	هذه الأداة موجودة في panel5	
bunifuTextBox1	BorderRadius	1
	FillColor	White
	ForeColor	ControlText
	LeftIcon	
	Size	200, 35
	Style	Material
	TextPlaceholder	Search for something...

[1] يمكنك ضبط خاصية السماح بتغيير حجم النافذة بعد استخدام بعض الأكواد لضبط مواقع وحجوم الأدوات الأخرى التي ستتأثر بتغيير وضعية النافذة WindowState، وبعد إضافة أزرار تكبير وتصغير النافذة.
[2] اسم الخاصية هو AllowObacityChangesWhileDragging.
[3] يمكنك ضبطها على True لا مشكلة.

<table>
<tr><td rowspan="7">bunifuTextBox2</td><td>BorderRadius</td><td>2</td></tr>
<tr><td>FillColor</td><td>White</td></tr>
<tr><td>ForeColor</td><td>ControlText</td></tr>
<tr><td>Size</td><td>200, 35</td></tr>
<tr><td>Style</td><td>Material</td></tr>
<tr><td>TextPlaceholder</td><td>Your first name</td></tr>
<tr><td colspan="2">بالمثل هناك ثلاث أدوات أخرى تختلف في الحجم
والخاصية الأخيرة فقط</td></tr>
<tr><td rowspan="10">button1</td><td>Cursor</td><td>Hand</td></tr>
<tr><td>Dock</td><td>Top</td></tr>
<tr><td>FlatAppearance.BorderSize</td><td>0</td></tr>
<tr><td>FlatStyle</td><td>Flat</td></tr>
<tr><td>Font</td><td>Segoe UI Light, 12pt</td></tr>
<tr><td>ForeColor</td><td>ControlText</td></tr>
<tr><td>Size</td><td>150, 90</td></tr>
<tr><td>Text</td><td>M y D a t a</td></tr>
<tr><td>TextImageRelation</td><td>ImageAboveText</td></tr>
<tr><td colspan="2">هناك أربع أدوات أخَر لها نفس خصائص هذه الأداة تقريبًا
هذه الأدوات كلها موجودة في القسم الأيسر</td></tr>
<tr><td rowspan="6">Button6</td><td>FlatAppearance.BorderSize</td><td>0</td></tr>
<tr><td>FlatStyle</td><td>Flat</td></tr>
<tr><td>Font</td><td>Segoe UI Light, 12pt</td></tr>
<tr><td>ForeColor</td><td>ControlText</td></tr>
<tr><td>Size</td><td>150, 90</td></tr>
<tr><td>Text</td><td>Hasan M. al-Fahl</td></tr>
</table>

TextBeforeImage	TextImageRelation	
من المفترض عند النقر على هذا الزر تظهر قائمة، سنتجاهل هذه النقطة.. هذه الأداة موجودة في القسم الأعلى من التطبيق		
White	BackColor	button7 & button8
DodgerBlue	FlatAppearance.BorderColor	
0	FlatAppearance.BorderSize	
Flat	FlatStyle	
Segoe UI Light, 12pt	Font	
DodgerBlue	ForeColor	
150, 40	Size	
Next step & Prev step	Text	
DodgerBlue	BackColor	button9
LightSlateGray	FlatAppearance.BorderColor	
0	FlatAppearance.BorderSize	
Flat	FlatStyle	
Segoe UI Light, 12pt	Font	
White	ForeColor	
150, 40	Size	
New Plan!	Text	
Fill	Dock	label1
Segoe UI Light, 15.75pt	Font	
White	ForeColor	

E N G 2 7	Text	
MiddleCenter	TextAlign	
هذه الأداة موجودة في القائمة panel3 الموجودة أعلى القسم الأيسر من التطبيق بقية أدوات العناوين بحجم خط 12 بنوع Segoe UI Light أو بحجم خط 9		
ControlDark	BorderColor	lineShape1
600	X1	
600	X2	
6	Y1	
54	Y2	
White	BackColor	panel1
Left	Dock	
150	Width	
هذه اللائحة في أسفل طبقة من طبقات البرنامج		
White	BackColor	panel2
Top	Dock	
60	Height	
DodgerBlue	BackColor	panel3
Top	Dock	
60	Height	
هذه اللائحة موجودة في اللائحة panel1		
DodgerBlue	BackColor	panel4
147, 422	Location	
3, 86	Size	

	هذا اللائحة موجودة في اللائحة panel1 وهي أعلى طبقة	
panel5	BackColor	White
	Location	214, 76
	Size	200, 200
panel6	BackColor	White
	Location	214, 301
	Size	200, 250
panel7	BackColor	White
	Location	439, 76
	Size	500, 150
panel8	BackColor	White
	Location	439, 246
	Size	500, 305
pictureBox1	SizeMode	Zoom
	هذه الأداة موجودة في panel6	
stepsProgress1	CurrentStep	3
	LineColor	DodgerBlue
	StepCurrent	
	StepDone	
	Steps	4
	StepUnDone	
	Width	462

اعتمد على هذا الكود:



```
using System;
using System.Windows.Forms;

namespace BunifuTest3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent(); stepsProgress1.CurrentStep = 3;
        }

        void ButtonsClick(object sender, EventArgs e)
        {
            Button b = (Button)sender; panel4.Top = b.Top  + 2;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            ButtonsClick(sender, e);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            ButtonsClick(sender, e);
        }

        private void button3_Click(object sender, EventArgs e)
        {
            ButtonsClick(sender, e);
        }

        private void button4_Click(object sender, EventArgs e)
        {
            ButtonsClick(sender, e);
        }

        private void button5_Click(object sender, EventArgs e)
        {
            ButtonsClick(sender, e);
        }

        private void button7_Click(object sender, EventArgs e)
        {
            stepsProgress1.CurrentStep += 1;
            if (stepsProgress1.CurrentStep > 2)
                label6.Text = "YOUR PLAN IS ALMOST DONE!";
        }

        private void button8_Click(object sender, EventArgs e)
        {
            stepsProgress1.CurrentStep -= 1;
            if (stepsProgress1.CurrentStep < 2)
                label6.Text = "YOUR PLAN IS AT ITS BEGGINING!";
        }
    }
}
```

بتشغيل التطبيق ستحصل على:

لاحظ بساطة النافذة وجمالية أدواتها؟ هل تذكر مشروع تطبيق إدارة الخدمات الذي صممناه بمنصة Guna.UI؟؟ ذكرنا وقتها أن بعض الناس – وقد حدث فعلًا – يعتقد أن تلك النافذة مصممة بلغات برمجة الويب، هل تصدّق أن أناسًا آخرين ظنوا نفس الأمر مع هذه النافذة!! للعين دورها أيضًا.. هل تذكر عندما ذكرنا أن التطبيق إذا كان جميلًا وأنيقًا حتى لو كان ضعيفًا كمضمون هو أفضل من ذلك القوي بمضمونه الضعيف بشكله؟؟؟

للمزيد، راجع هذه الروابط.[1]

---

[1] روابط مفيدة للاستفادة منها في التصميم:
https://medium.com/bunifuframework/how-i-designed-a-pc-maintenance-ui-ux-dashboard-e861f4242aef
https://bunifuframework.com/demos/#!
https://medium.com/bunifuframework/inspirations/home
https://medium.com/bunifuframework

الفصل الحادي عشر – منصة DevExpress

تعمّدتُ تأخير هذه المنصة وجعلتها في فصل منفصل، فهي أغنى وأقوى وأجمل وأتقن المنصات!! تطبيقات المنصة عديدة وكثيرة، فيمكن استخدامها في تطبيقات سطح المكتب وتطبيقات الويب وتطبيقات الموبايل، وأدواتها مصممة باحترافية كبيرة، فاحتمال مواجهتك لأخطاء برمجية أثناء استخدامك لأدوات المنصة ضئيل جدًا ويكاد يكون معدومًا (على عكس بعض المنصات السابقة).

من المنصات الكبيرة أيضًا منصة Telerik، ومنصة Syncfusion، والتي تعطيك رخصة مجانية إذا كان دخلك السنوي محدودًا ووعدد أفراد شركتك لا يتجاوز الخمس مبرمجين (لا أظن أن دخلك يتجاوز المليون دولار، ويعمل في شركتك غيرك، إلا إذا كنت من دبي!).

تعطيك هذه المنصة عشرات الأدوات البرمجية والتي يمكن استخدامها بتنسيقات مختلفة كثيرة، وتدعم التقارير ولوحات التحكم بشكل كبير جدًا، كما تزودك بخدمات APIs للتعامل مع ملفات Office، والمزيد!!

مدخل إلى منصة DevExpress

المتطلبات الأساسية Prerequisites

الجدول التالي يحدد نسخ الدوت نت المطلوبة لتشغيل نسخ منصة DevExpress المختلفة، حتى نسخة 19.2 (آخر نسخة حتى فترة كتابة هذا الفصل):

DevExpress Version	.NET 2.0	.NET 3.5	.NET 4.0	.NET 4.5	.NET 4.5.2	.NET 4.6	.NET 4.7	.NET 4.8
v19.2 (current)	✗	✗	✗	✗	✓	✓	✓	✓
Other Versions								
v18.2-v19.1	✗	✗	✗	✗	✓	✓	✓	✓
v16.2-v18.1	✗	✗	✓	✓	✓	✓	✓	✗
v14.2-v16.1	✗	✗	✓	✓	✓	✓	✗	✗
v13.1-v14.1	✗	✗	✓	✓	✓	✗	✗	✗
v12.1-v12.2	✗	✓	✓	✓	✓	✗	✗	✗
v11.2	✗	✓	✓	✗	✗	✗	✗	✗
v10.1-v11.1	✓	✓	✓	✗	✗	✗	✗	✗
v9.3	✓	✓	✗	✗	✗	✗	✗	✗

أما بخصوص نسخ الفيجوال ستوديو:

DevExpress Version	Visual Studio 2005	Visual Studio 2008	Visual Studio 2010	Visual Studio 2012	Visual Studio 2013	Visual Studio 2015	Visual Studio 2017	Visual Studio 2019
v19.2 (current)	✗	✗	✗	✓	✓	✓	✓	✓
Other Versions								
v18.2	✗	✗	✗	✓	✓	✓	✓	✓*
v18.1	✗	✗	✓	✓	✓	✓	✓	✓**
v16.1.11-v17.2	✗	✗	✓	✓	✓	✓	✓	✗
v14.2-v16.1.10	✗	✗	✓	✓	✓	✓	✗	✗
v12.2.15-v14.1	✗	✗	✓	✓	✓	✗	✗	✗
v12.1-v12.2.14	✗	✓	✓	✓	✗	✗	✗	✗
v11.2	✗	✓	✓	✗	✗	✗	✗	✗
v10.1-v11.1	✓	✓	✓	✗	✗	✗	✗	✗
v9.3	✓	✓	✗	✗	✗	✗	✗	✗

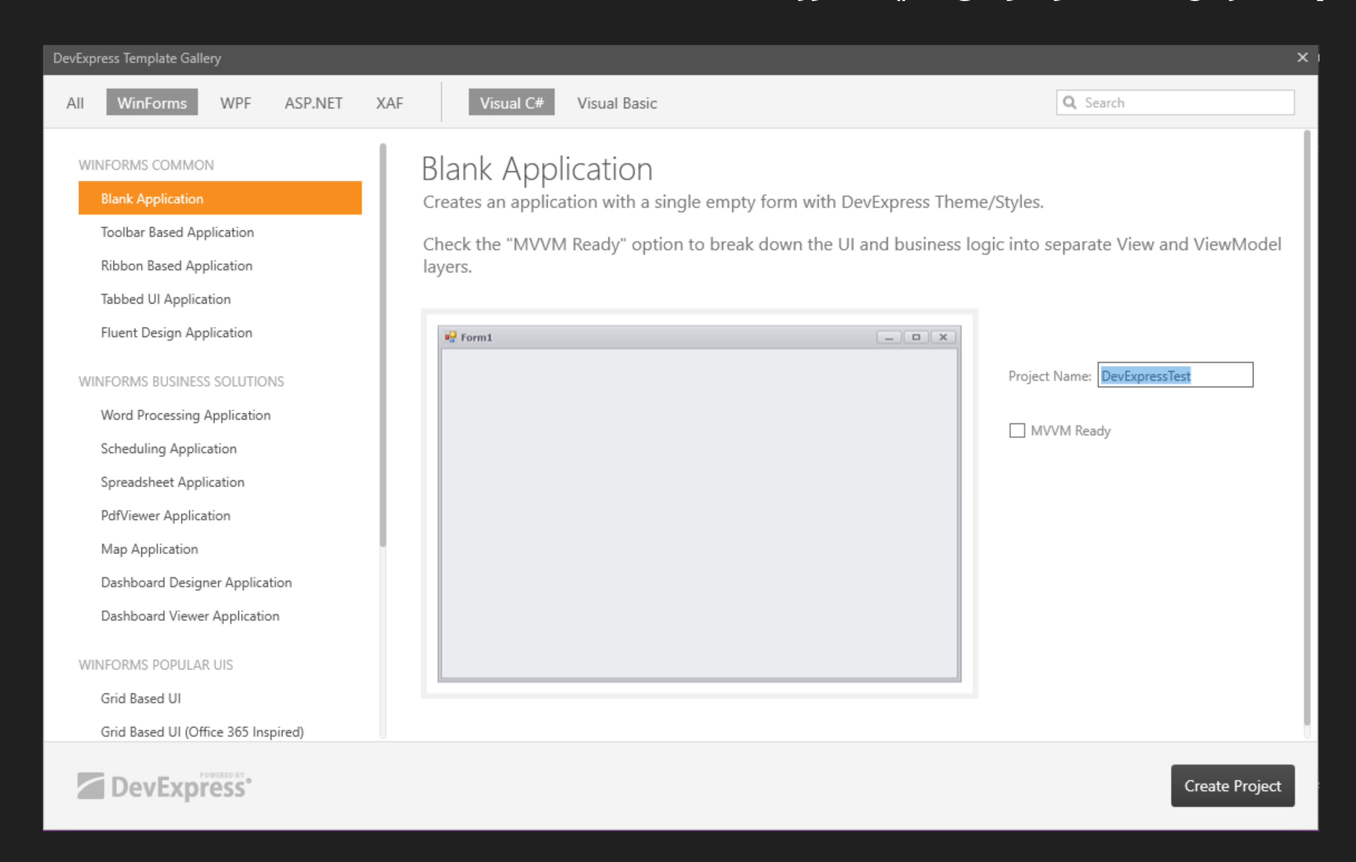
* بدءًا من النسخة 18.2.7، ** بدءًا من النسخة 18.1.11. يمكنك التعرف على المزيد من هنا [1].

مستعرض نماذج ديف إكسبرس

بعد تثبيت نسخة ديف إكسبرس في جهازك ستحصل على مستعرض نماذج (أو قوالب) ديف إكسبرس DevExpress Template Gallery، وفيه يمكنك اختيار شكل عام لتطبيقك، كما يمكنك استخدام منصة ديف إكسبرس دون الاعتماد على مستعرض النماذج وذلك من خلال إنشاء مشروع نوافذ عادي.

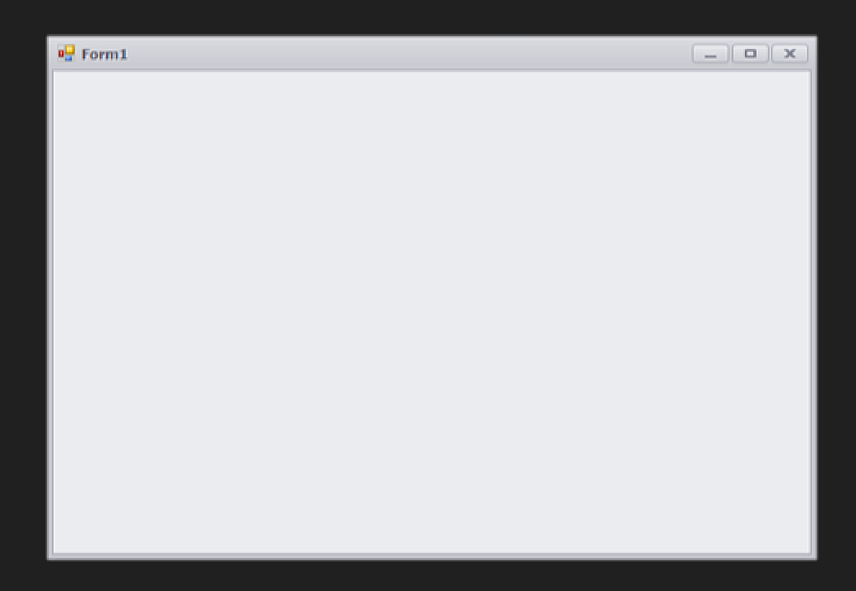
[1] متطلبات تشغيل منصة ديف إكسبرس
https://documentation.devexpress.com/WindowsForms/8092/Prerequisites

يمكنك من خلال مستعرض النماذج Template Gallery إنشاء أنواع مختلفة لنماذج ديف إكسبرس، كما هو موضح في الصورة التالية:

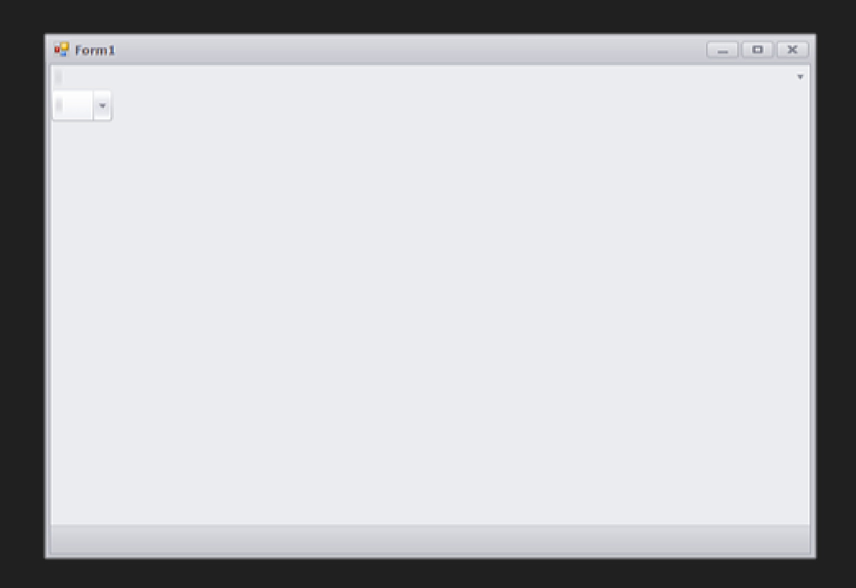


الصورة السابقة فيها المشاريع الجاهزة التالية:

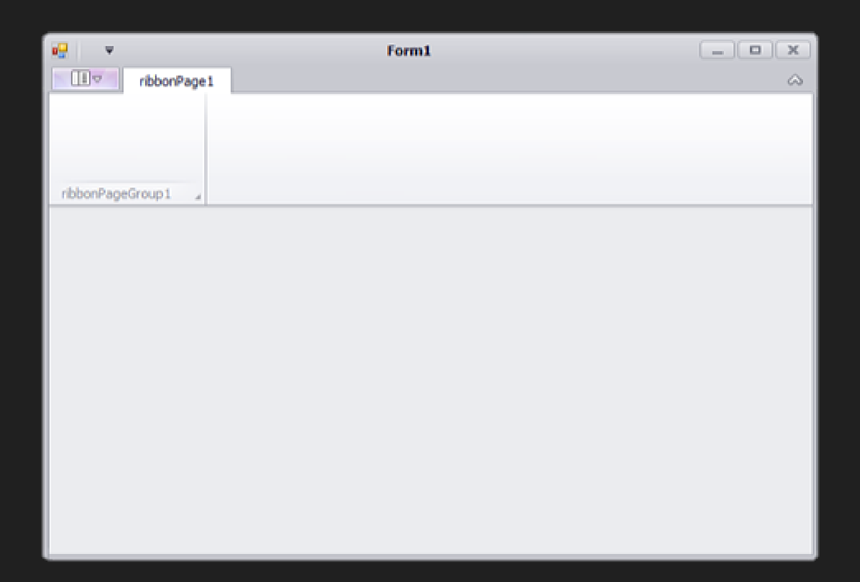
التطبيق الفارغ Blank Application: وهو نموذج من النوع XtraForm، وهو أول شكل من أشكال نماذج ديف إكسبرس. (يكافئ النموذج Form العادي في منصة دوت نت).



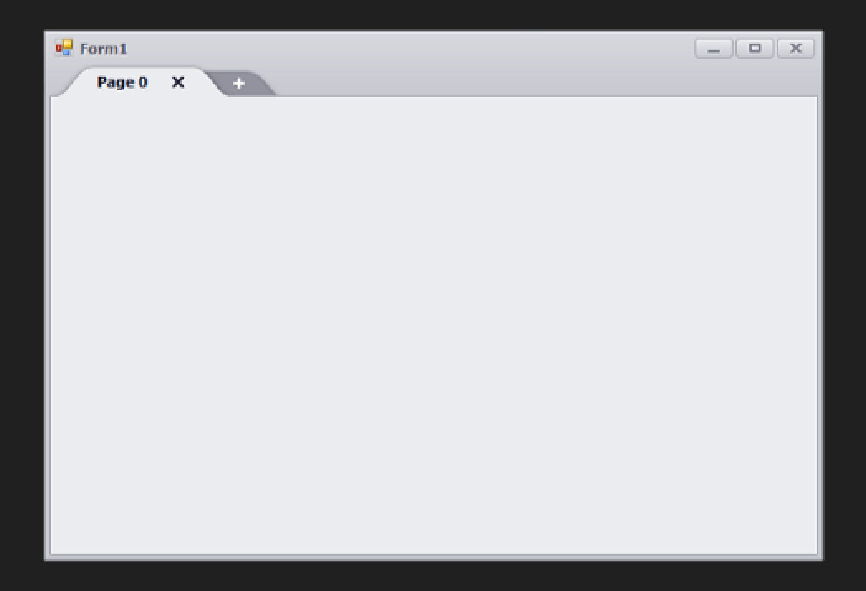
التطبيق المبني على القوائم Toolbar Based Application: وهو نفس التطبيق السابق لكن مع قوائم.



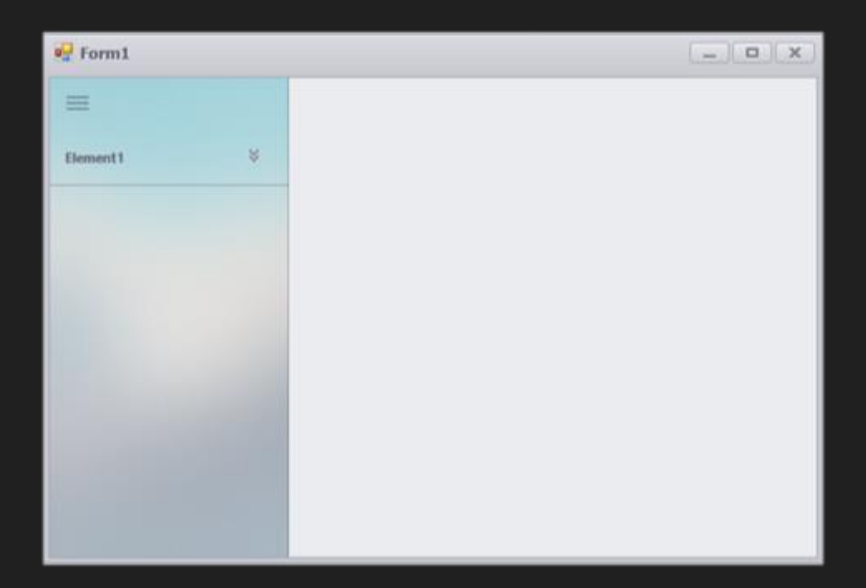
التطبيق المبني على شريط Ribbon Based Application: وهو نموذج يحوي شريطًا أعلاه فيه مجموعة تبويبات، في كل تبويبة عدة مجموعات، في كل مجموعة أمر أو أكثر. وأشهر البرامج المشهورة المبنية على الأشرطة Ribbons هي برامج الأوفيس.



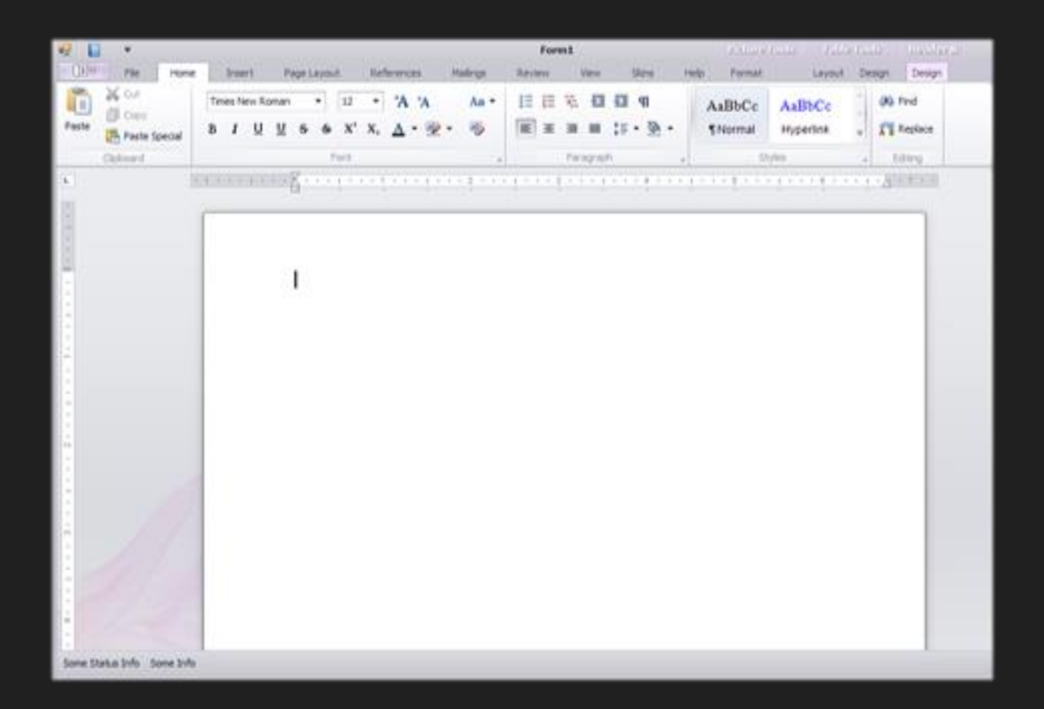
التطبيق المبني على تبويبات Tabbed Based Application: وهو مشابه للتطبيق السابق إلا أنه لا يحوي مجموعات، فقط تبويبات. ويوازي في أدوات ويندوز القياسية أداة TabControl. أشهر البرامج التي تستخدم هذا النوع من النماذج هي متصفحات الإنترنت.



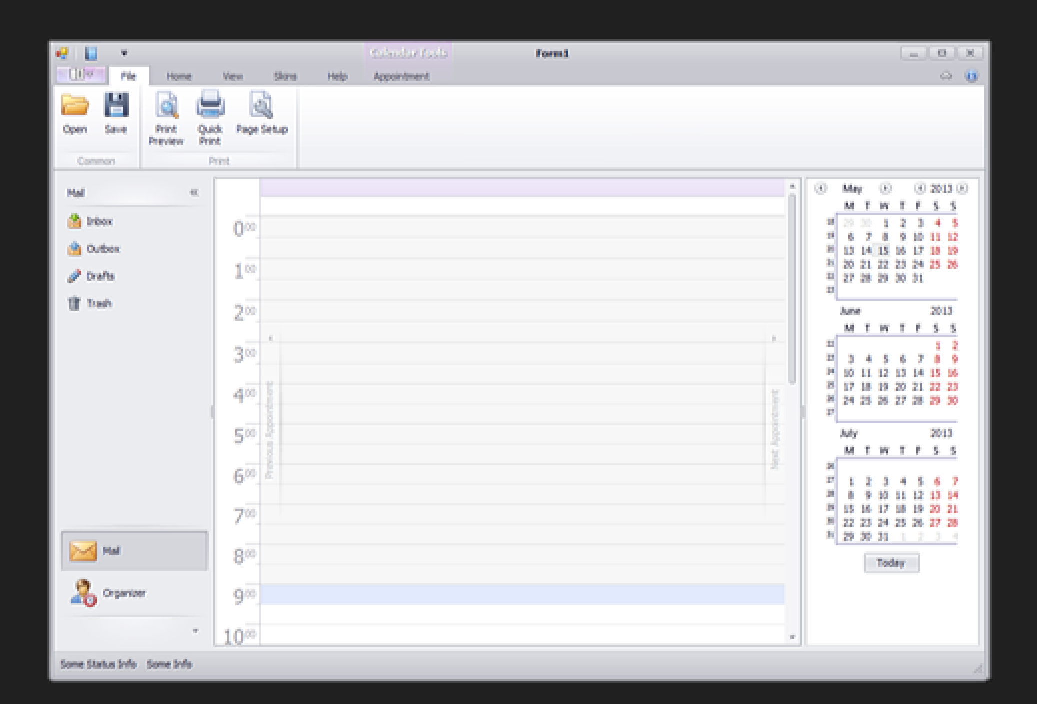
التطبيق مرن التصميم Fluent Design Application: وهو نموذج معتمد على وجود قائمة جانبية، مثل إعدادات ويندوز 10.



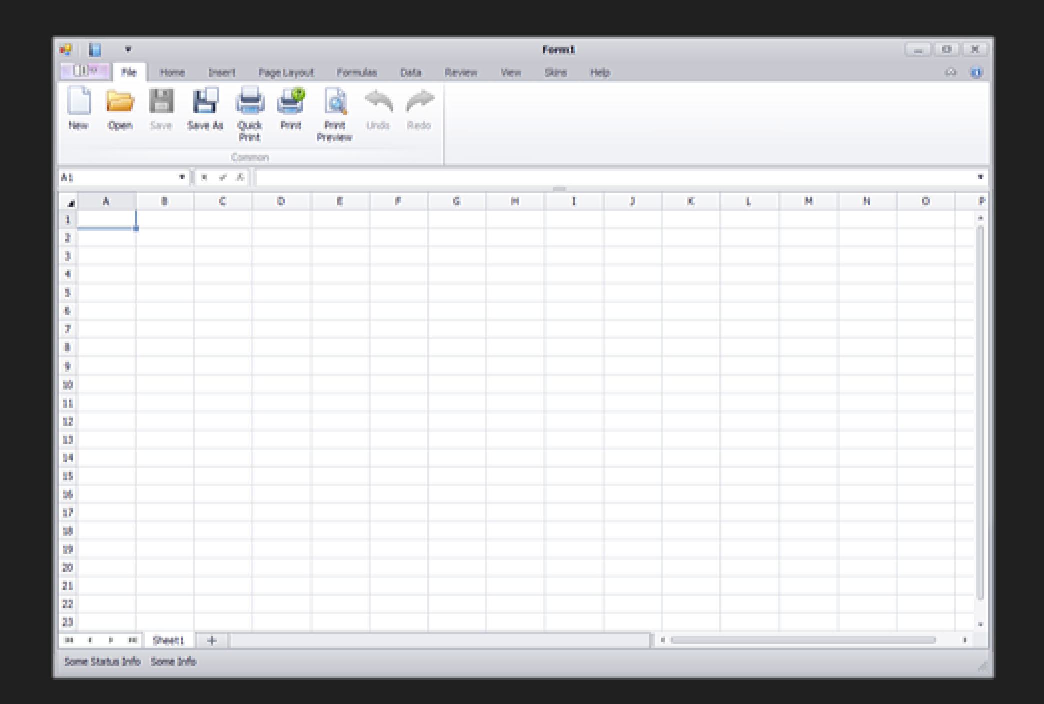
تطبيق معالجة ملفات الوورد Word Processing Application: وهو نموذج يعطيك إمكانيات كبيرة في معالجة النصوص والتعامل مع ملفات الوورد بمزايا كثيرة.



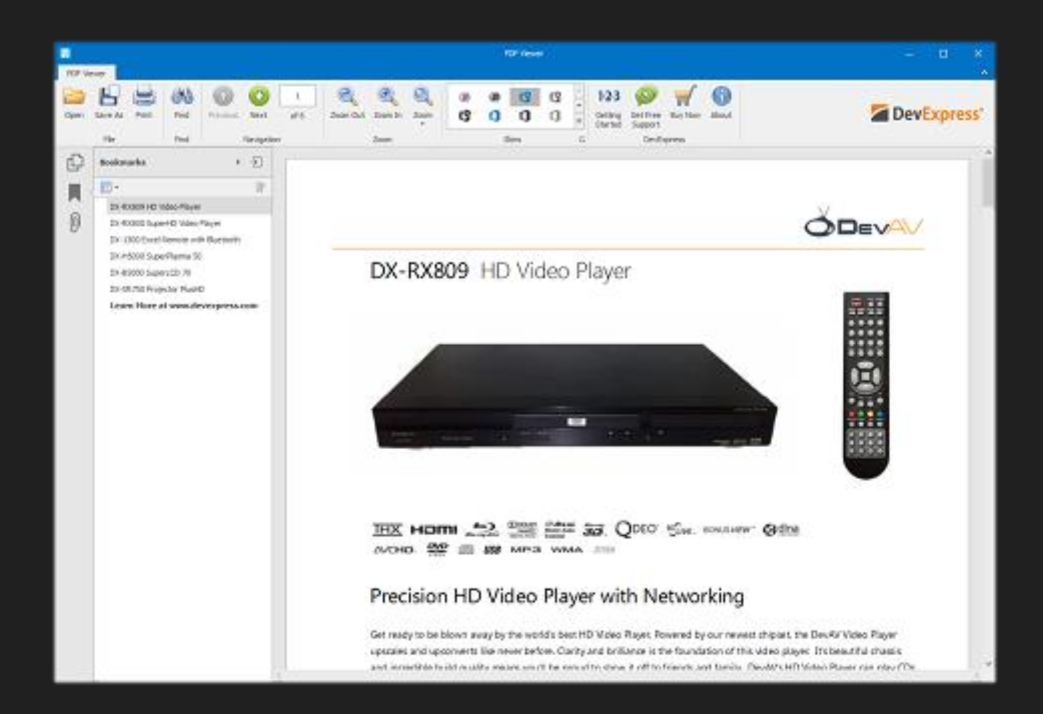
تطبيق جدولة المهام Scheduling Application: وهو نموذج يعطيك إمكانيات كبيرة في جدولة المهام وإدارتها.



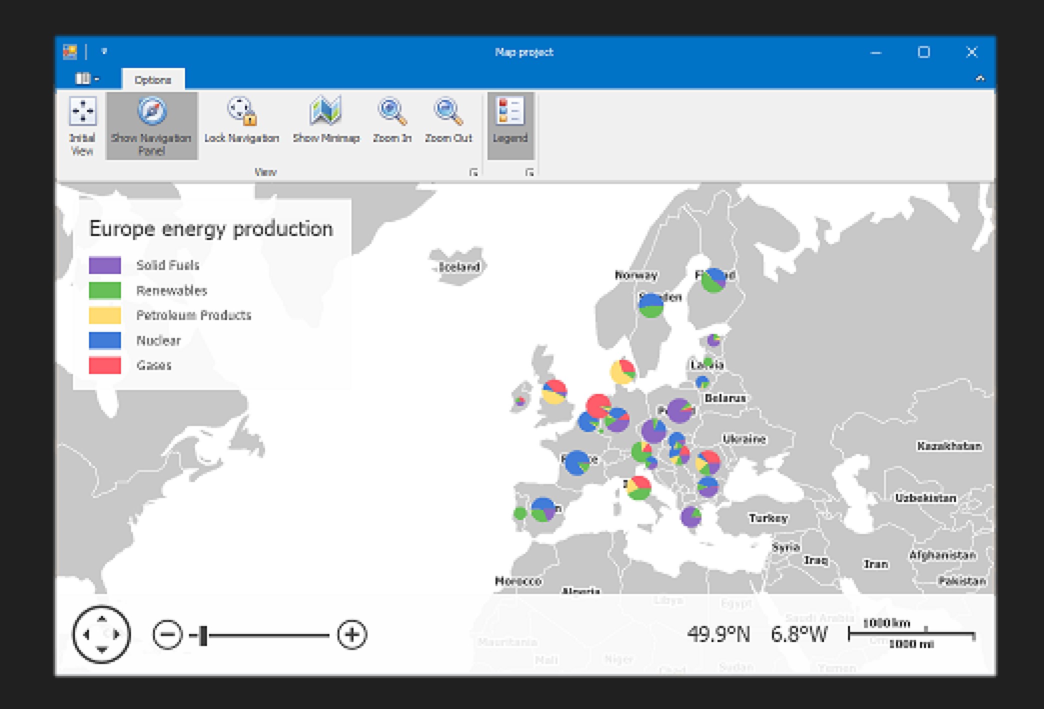
تطبيق جداول البيانات Spreadsheet Application: وهو نموذج يعطيك إمكانيات التعامل مع ملفات الإكسل، بمزايا كثيرة.



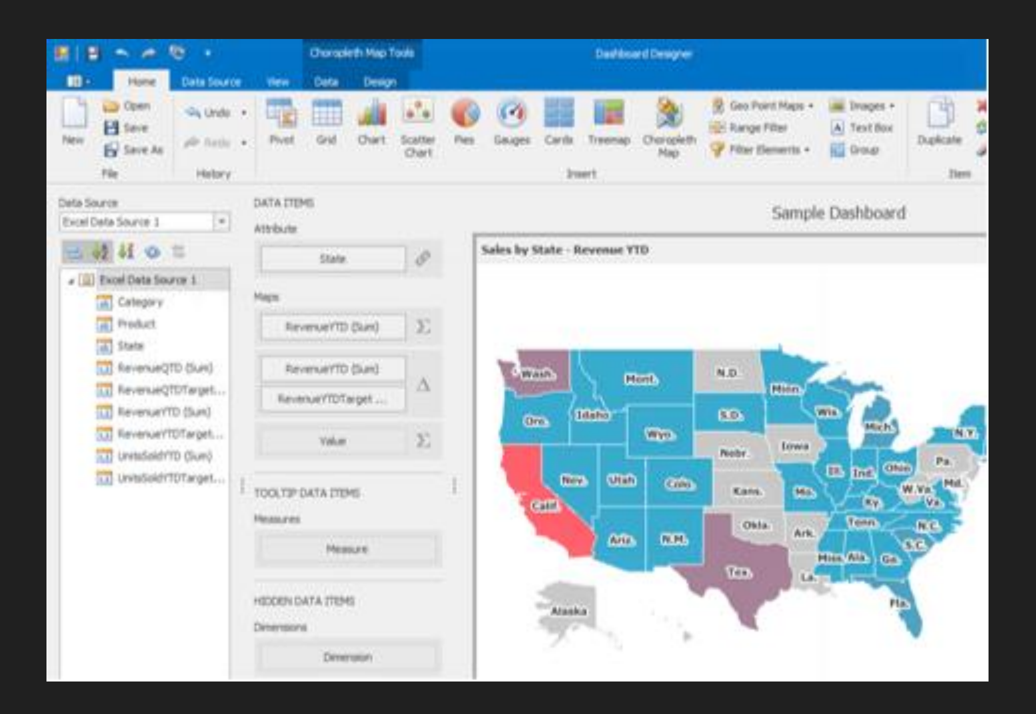
تطبيق عارض ملفات PdfViewer Application pdf: سلامة فهمك 😃.



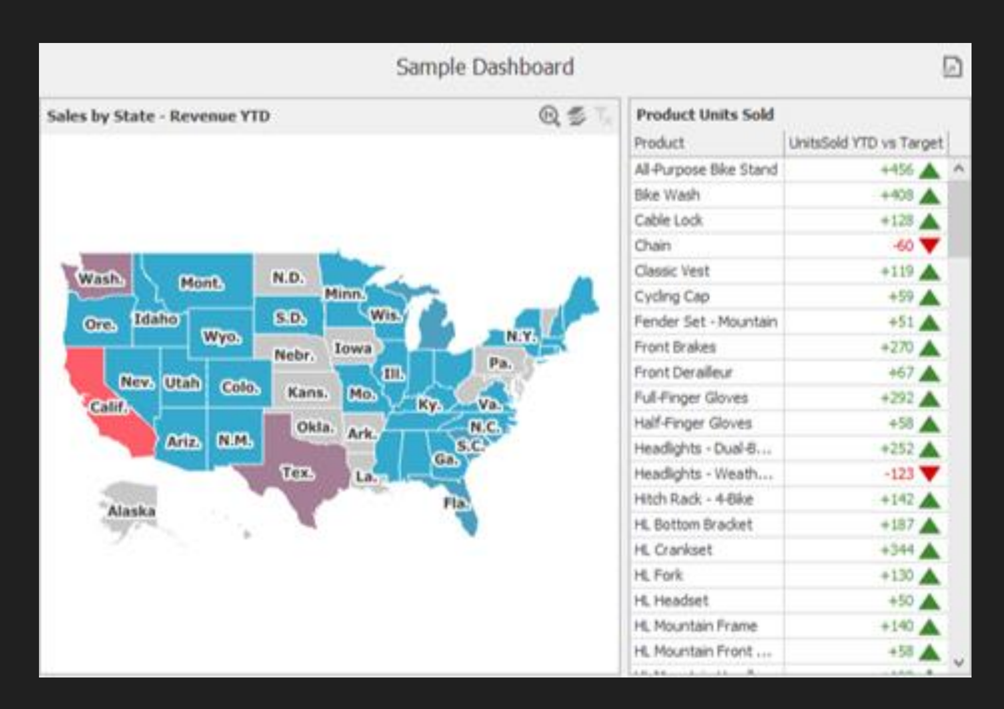
تطبيق الخرائط Map Application: هو نموذج يعطيك إمكانية التعامل مع الخرائط.



تطبيق مصمم لوحة التحكم Dashboard Designer Application: وفيه يمكنك تصميم لوحة تحكم قوية.



تطبيق عارض لوحة تحكم Dashboard Viewer Application: وهو نموذج يعرض لوحات التحكم.



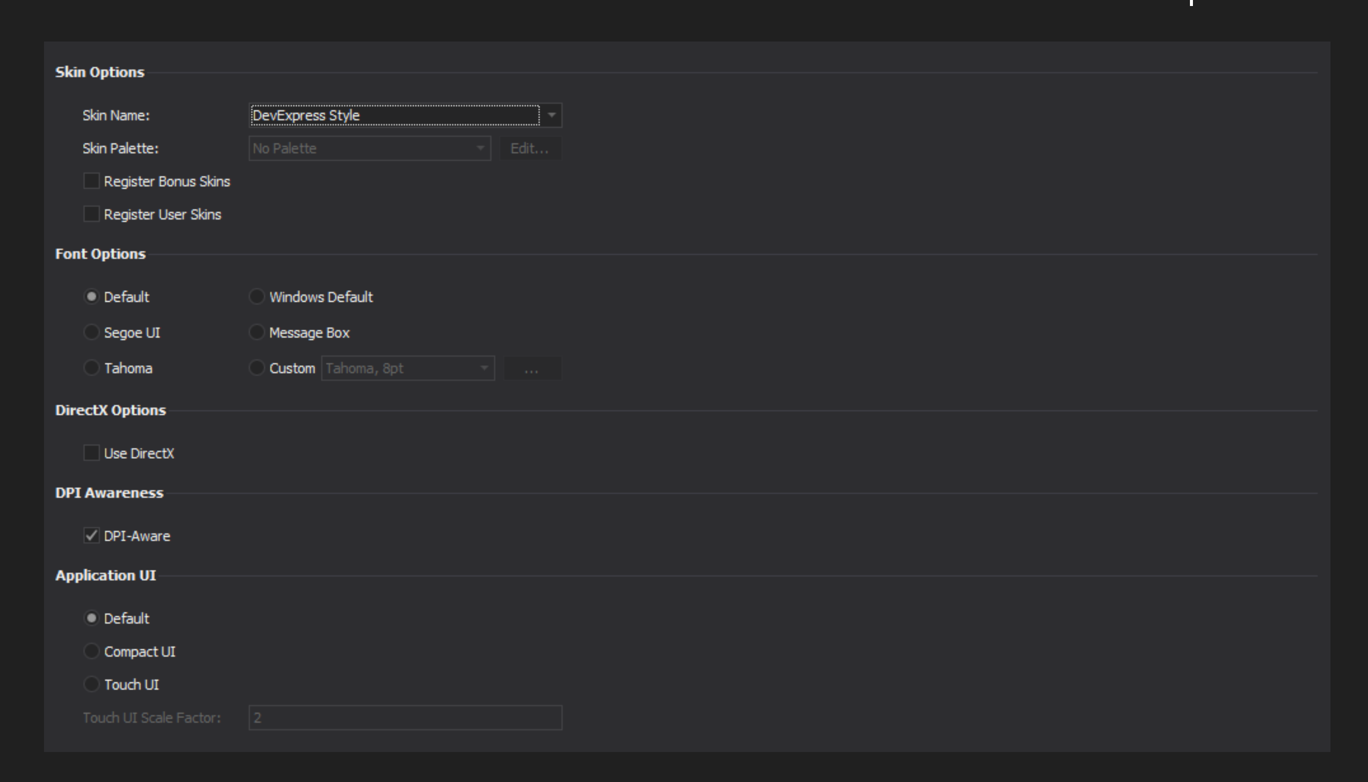
والمزيد!

وعلى اعتبار أن النظام بأكمله سيُبنى على أدوات ونماذج ديف إكسبرس، فجميع واجهات البرنامج ستكون واحدة مما سبق (وغيرها)، إلا أن الواجهة الأساسية هي أول واجهة تختارها ضمن المشروع (أو الواجهة التي ستضبط المشروع أن يعمل عليها). أي أن المشروع يمكن أن يحوي أكثر من واجهة مما سبق، فليس بالضرورة أن يحوي واجهة واحدة أو نوعًا واحدًا من النماذج.

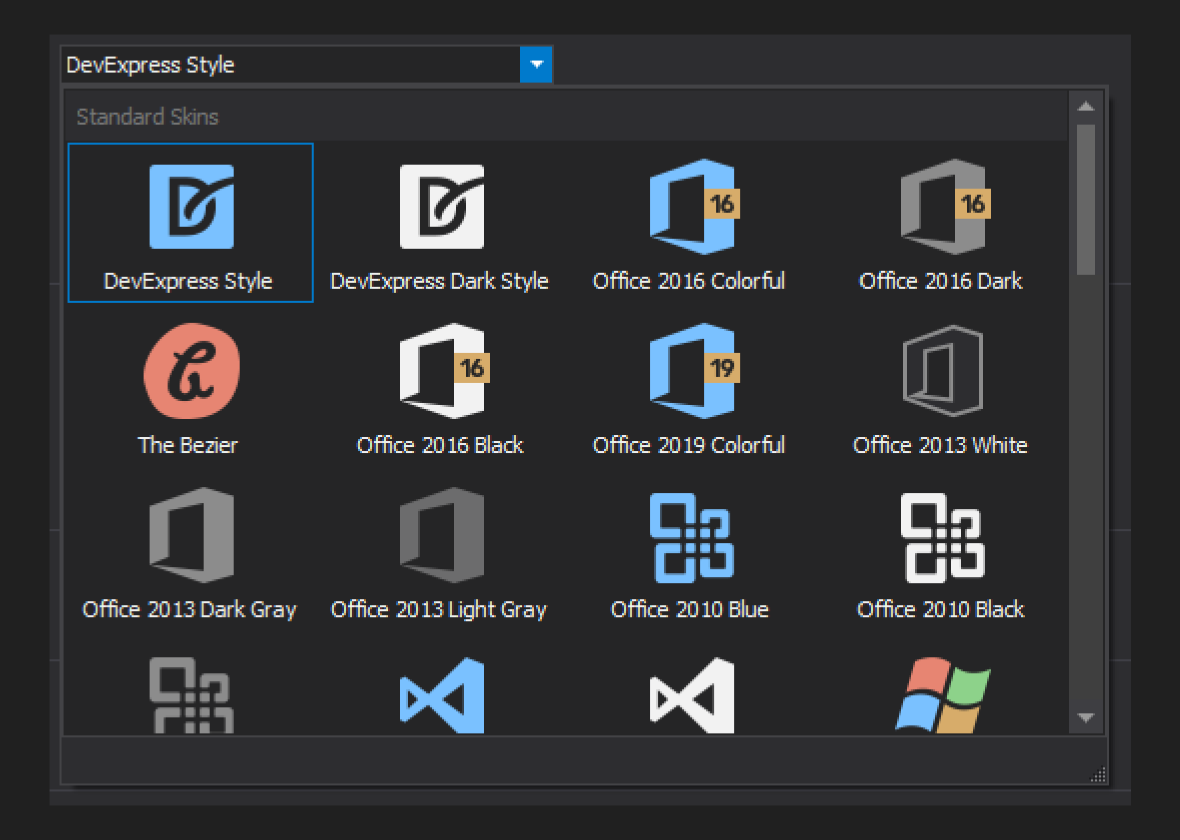
إعدادات مشروع DevExpress، ومظهره

عند ضبط إعدادات المشروع العامة فإن مظهر الأدوات والنوافذ يتم تغييرها، إلا أنه لا زال هناك فرصة لتغييرها لاحقًا.

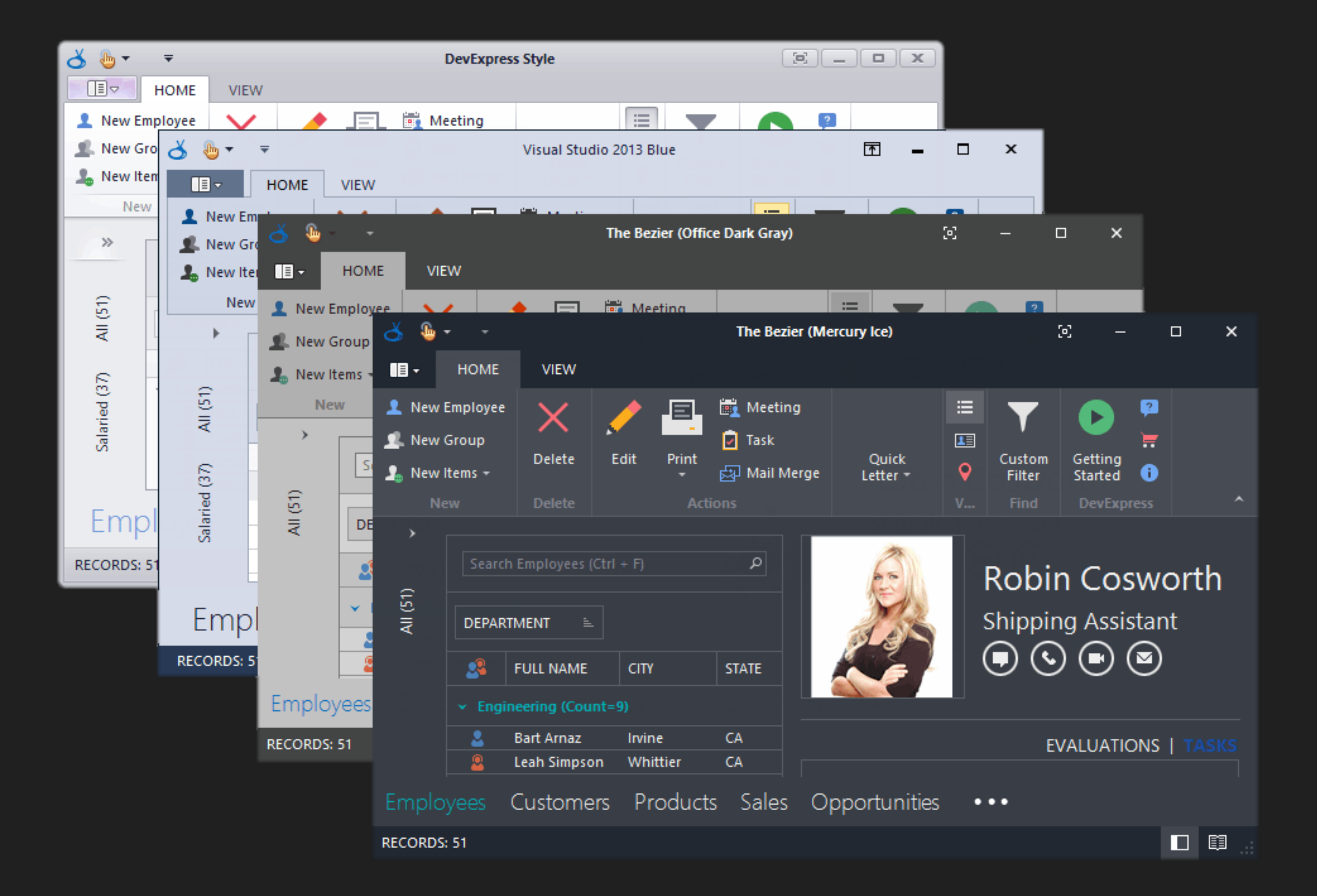
يمكنك ضبط مظهر مشروعك العام من خلال DevExpress Project Settings، والتي يمكنك الوصول إليها عبر القائمة DevExpress ثم WinFroms Controls:



يمكنك من خلال صفحة إعدادات المشروع ضبط الخط العام لتطبيقك وأشياء أخرى، لعل أهمها مظهره. يمكن أن يأخذ مظهر المشروع Skin كثيرًا من الأشكال، وهي لبرامج مشهورة وغير مشهورة:



للخاصية LookAndFeel دور كبير في مظهر تطبيقك، فهي تغير مظهره ومظهر أدواته، لاحظ نفس التطبيق مع سمات Themes مختلفة:

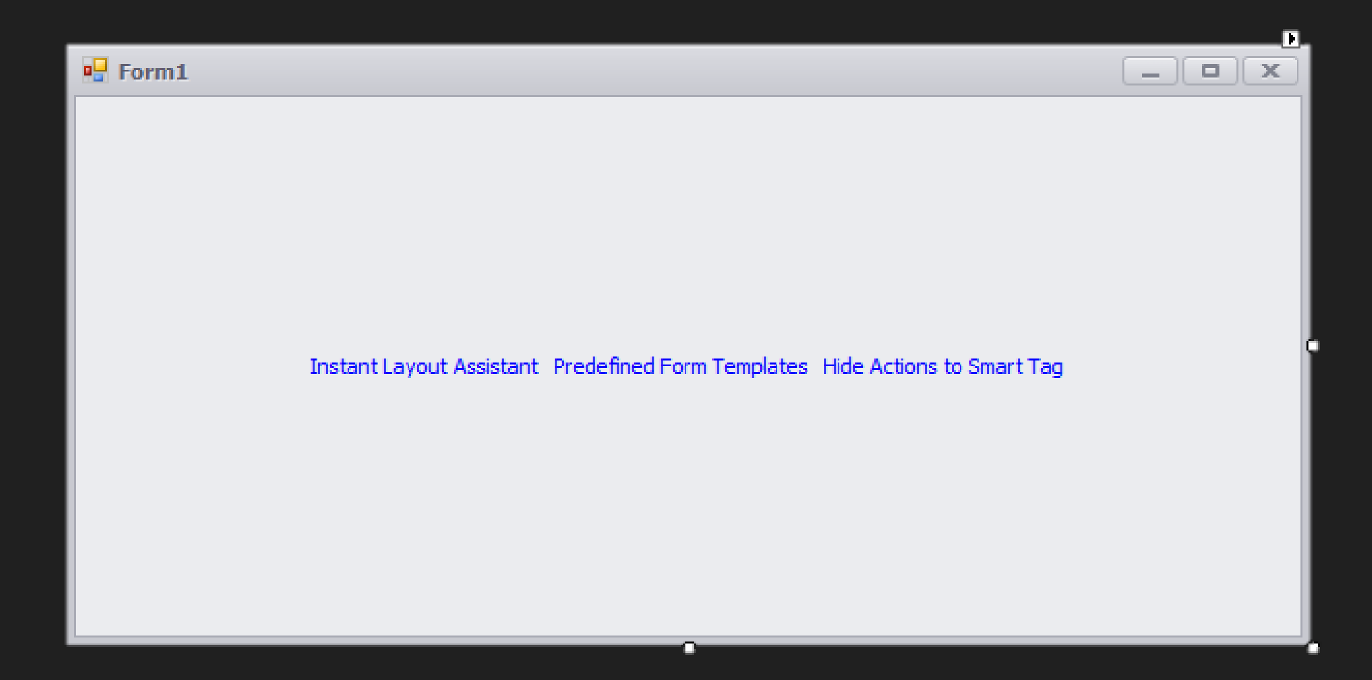


لتغيير مظهر التطبيق Skin من خلال خاصية LookAndFeel عليك ضبط خاصية UseDefaultLookAndFeel على القيمة False، ثم ضبط الخاصية SkinName كما ترغب. أو يمكنك استخدام الأداة DefaultLookAndFeel من صندوق الأدوات ToolBox، ومن خلالها يمكنك تعديل المظهر الافتراضي.

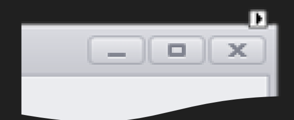
أما إذا أردت تغيير مظهر التطبيق بالكامل – كما سنرى لاحقًا من خلال أدوات معينة – فاضبط الخاصية المذكورة على القيمة True، وذلك لجميع الأدوات التي ترغب أن يتغير مظهرها مع التطبيق.

النموذج الافتراضي XtraForm

أضف نموذجًا فارغًا من نماذج DevExpress، من مستعرض نماذج ديف إكسبرس، لتحصل على النموذج الافتراضي الموضّح بالشكل:



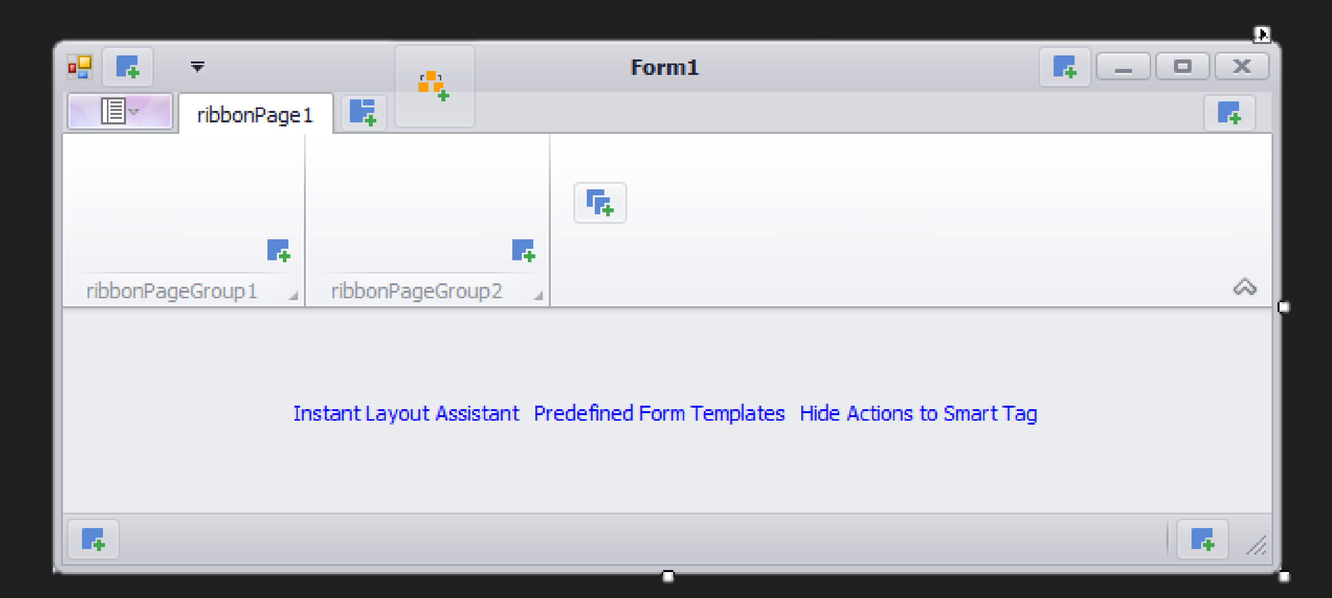
ومن الملاحظ أن حجم النافذة الافتراضي غير الحجم الذي اعتدنا عليه، كما أنه توجد ثلاثة عناوين تشعبية LinkLabels أولها من اليسار تقوم بتقسيم النافذة إلى أقسام وتعطيك عدة إمكانيات في كل قسم لتقوم بتنسيقه بنقرات قليلة، والوسطى تعطيك قوالب جاهزة يمكنك تطبيقها على نافذتك، والأخيرة تزيل هذه العناوين التشعبية الثلاثة وتنقلها لقسم الأوامر المميزة Smart Tag (السهم الصغير الموجود أعلى يمين التصميم).



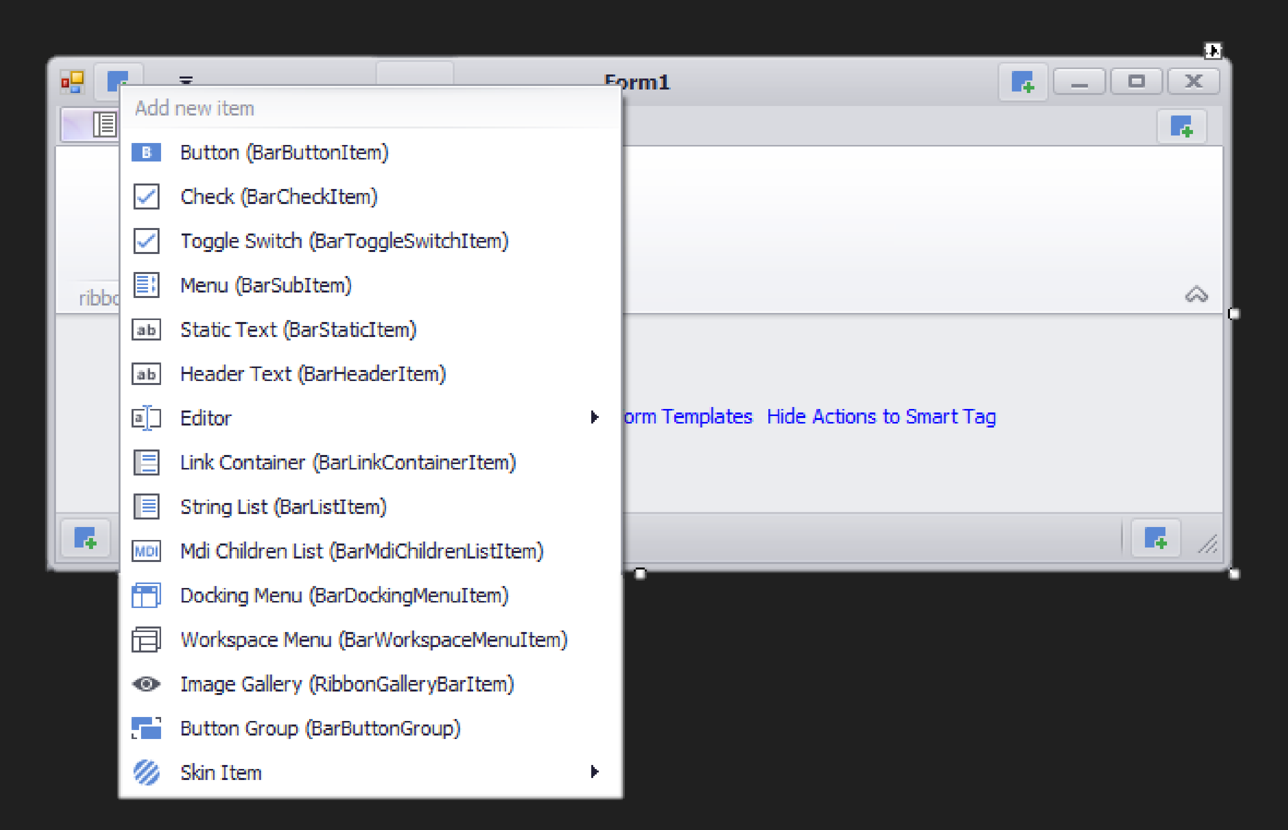
يكافئ هذا النموذج النموذج الافتراضي في ويندوز، ويزيد عليه ببعض الخصائص، وبإمكانية تنسيق الأدوات ضمنه بشكل أفضل وأسهل.

نموذج الأشرطة RibbonForm

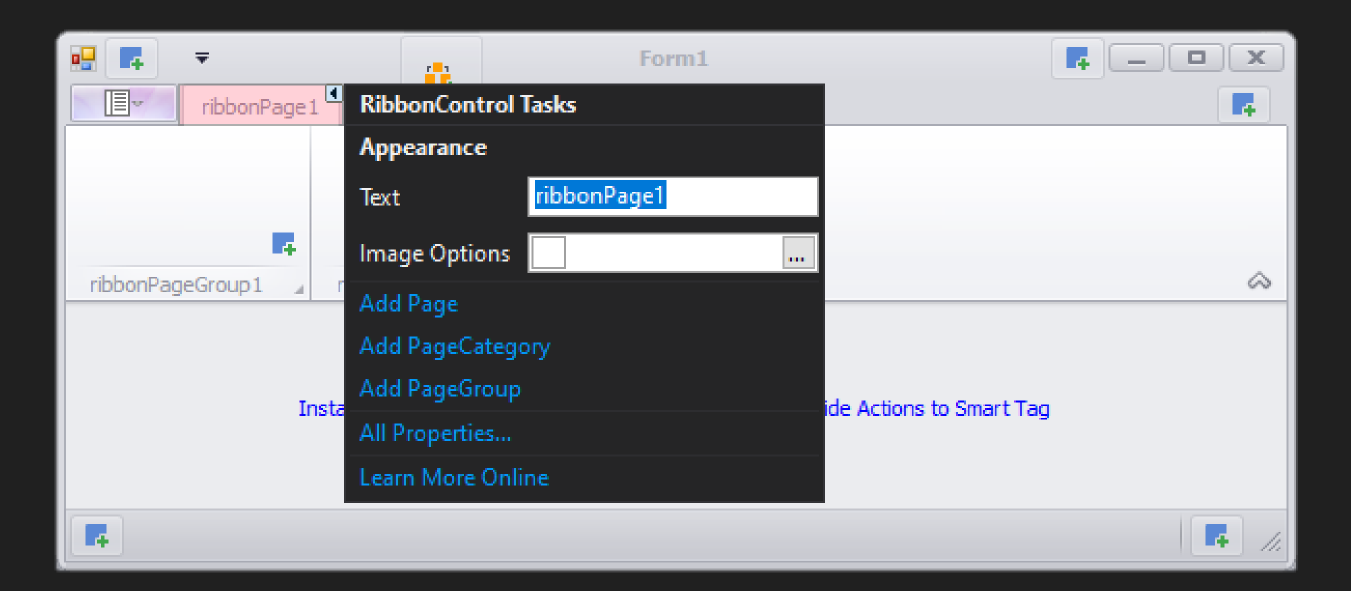
يمكنك من خلال الأوامر المميزة Smart Tag تحويل النموذج الافتراضي إلى نموذج أشرطة، كما يمكنك إضافة نموذج أشرطة من مستعرض نماذج ديف إكسبرس:



بالنقر على أحد الأزرار التي تحمل الصورة يمكنك إضافة بعض الأدوات:

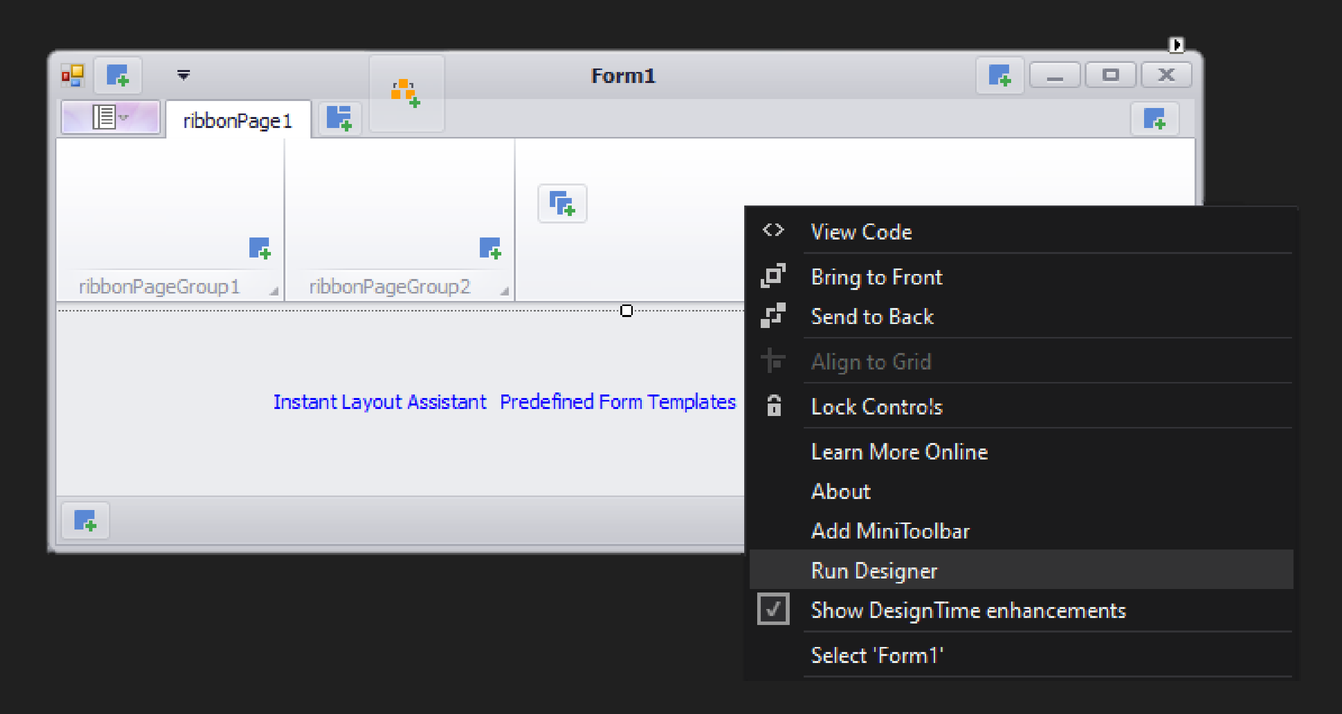


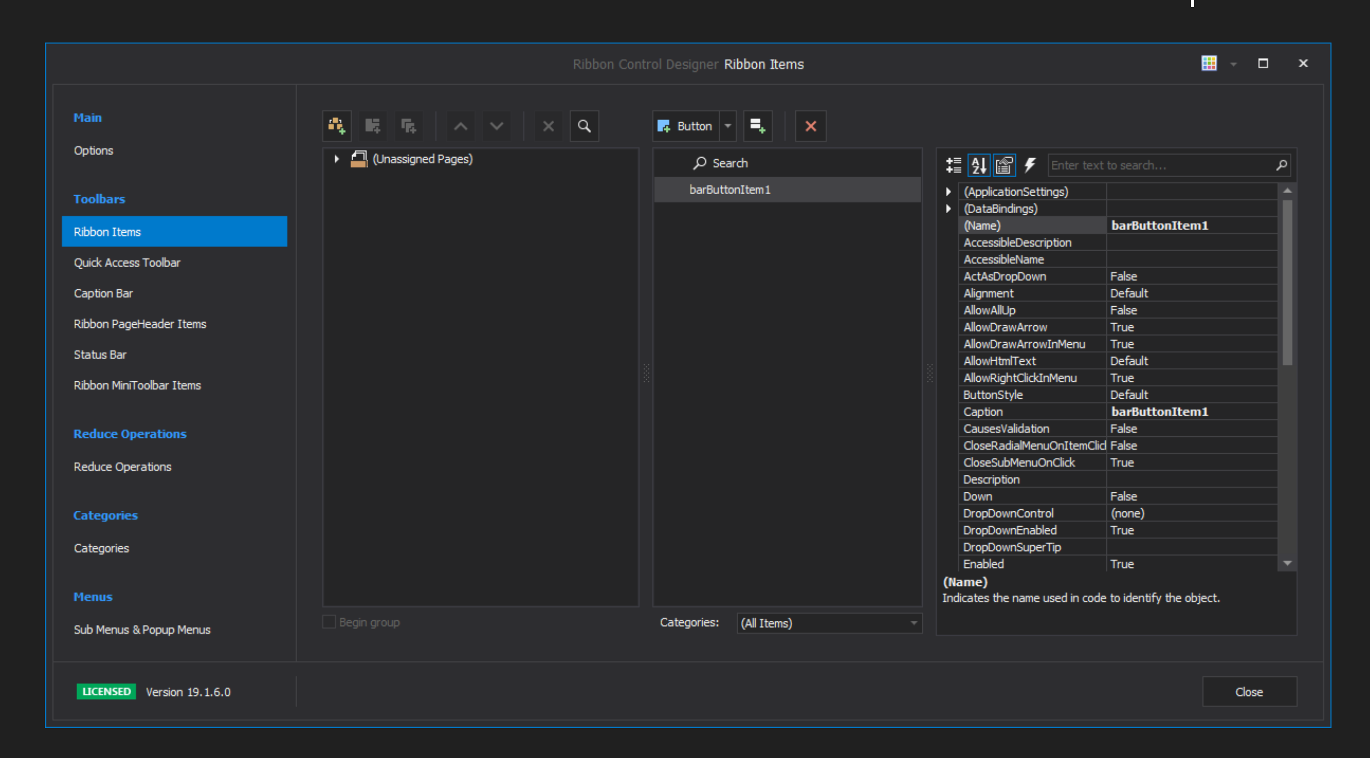
بالنقر على يمكنك إضافة قسم Category جديد. أما و فيمكنك من خلالها إضافة صفحات جديدة ومجموعات جديدة. كما يمكنك إضافة هذه الأشياء من خلال قائمة المهام الخاصة بصفحات النافذة:



كما يمكنك ذلك بالنقر بالزر الأيمن على أي صفحة من صفحات النافذة.

الأكثر من ذلك، يمكنك تصميم الشريط Ribbon من خلال مصم أداة الشريط Ribbon Control Designer، وذلك بالنقر بالزر الأيمن عليه ثم Run Designer:





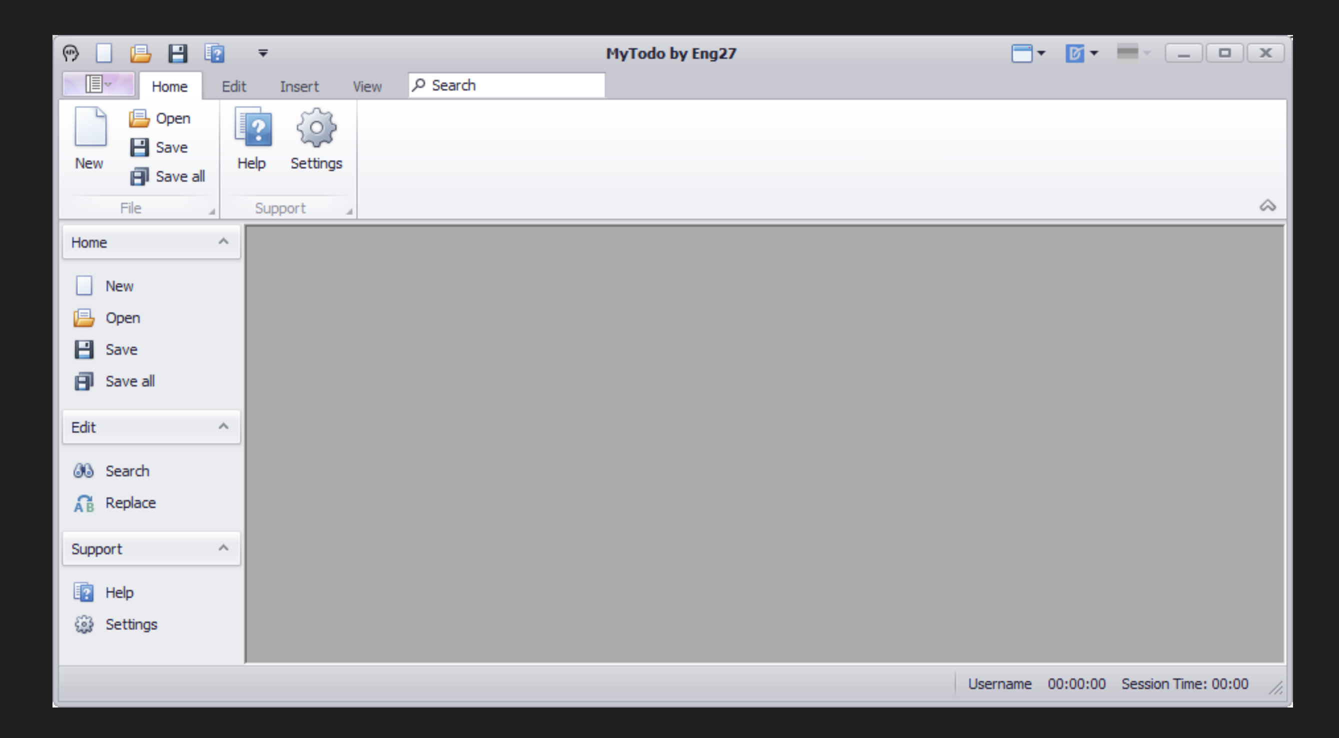
يُقسم المصمم إلى أربعة أقسام، الأول – من اليسار – يحوي أقسام الشريط المختلفة، والثاني يمثل ما هو موجود فعلًا في الأداة، والثالث فيه أدوات يمكنك تصميمها بشكل معين واستخدامها بأكثر من موضع (إن لم تضفها للقسم الثاني فلا وظيفة لها في الأداة المختارة)، والأخير يمكّنك من ضبط خصائص الأدوات في القسمين الثاني والثالث.

هذه الأداة تعطيك إمكانية تصميم المشروع بالكامل، فهي أشبه للمصمم designer بالنسبة لمشاريع WinForms التقليدية.

كما يمكنك القيام بكل هذا من خلال الأزرار ، و ، و التي ذكرناها سابقًا.

الصورة التالية فيها نموذج بسيط يحوي بعض الأقسام الشائعة في نماذج ويندوز، التي يمكنك إنشاؤها من خلال منصة DevExpress، وهي كما يلي:

- في القسم العلوي شريط العنوان TitleBar، وفيه شريط الوصول السريع QAT.
- تحته مباشرة صفحات الشريط Ribbon tabs، وفيها أيضًا الزر Application Button.
- في القسم السفلي شريط الحالة StatusBar.
- في القسم الأيسر شريط التنقل Nav Bar.
- في المنتصف، منطقة العمل.



المشروع الأول، مشروع إدارة مهام

ما سنقوم به – في الفقرات القادمة – هو مشروع بسيط لإيضاح إمكانيات عديدة يمكنك الوصول إليها في التطبيقات المبنية على الشرائط. هذا المشروع هو تطبيق يدير وينظم المهام. يعطيك المشروع إمكانية إضافة مهام وتعديلها وعرضها، وإنشاء "مشاريع"، حيث أن المشروع هو عبارة عن مجموعة من المهام، كما يعطيك إمكانية إضافة تفاصيل كثيرة لكل مهمة. ولإضافة إمكانية التعامل مع الملفات (حفظ، حفظ كـ، فتح، ...) سنحفظ المهام والمشاريع ضمن ملفات.

إن تخزين البيانات من الأفضل أن يكون ضمن قاعدة بيانات وليس ضمن ملفات. وكنتيجة لذلك فإن بعض العمليات مثل البحث والمقارنة بين البيانات ستكون محدودة ضمن هذا المشروع، إذ إن عمليات البحث ضمن الملفات أبطأ وأقل كفاءة من عمليات البحث ضمن قواعد البيانات.

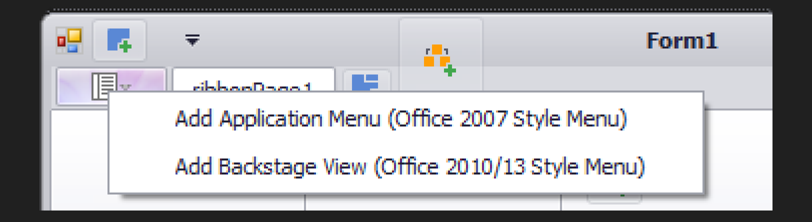
بيانات المشروع ستُخزَّن ضمن ملفات بصيغتين مختلفتين، مثلا: tsk (Task Document) وprj (Project File)، والغاية من تخزينه بصيغتين مختلفتين هي ليستطيع البرنامج التمييز بين الملفات المختلفة، وبالتالي التعامل مع الملفات على أساس نوعها (لكل نوع من الملفات أوامر مختلفة، وقد توجود أوامر مشتركة). وهذا اقتراح لا أكثر إن كنت ترغب بإكمال المشروع وتكويده (أي أنه يمكنك مثلًا إيجاد نظام ملفات خاص بك، بصيغة ملفات واحدة مثلًا، بحيث يكون محتوى الملف دالًا عليه هل هو مشروع Project أو مهمة Todo). كما يمكنك التعامل مع قواعد البيانات لهذا الأمر كما ذكرنا سابقًا، ولكنا – كما قلنا – سنخصص المشروع للتعامل مع الملفات لنضيف للمشروع الأوامر القياسية للتعامل مع الملفات.

ضع في ذهنك أن الفقرات التالية تشرح لك كيفية تصميم المشروع اعتمادًا على منصة ديف إكسبريس، أي أننا لن نناقش – مثلا – الطرق المثلى لتصميم المشاريع، أو تكويد المشروع حتى لو وضعنا بعض الأكواد هنا أو هناك، فمشروع مثل هذا – ولو كان بسيطًا – يحتاج لكتاب منفصل (أو كتيّب) ليشرحه تصميمًا وتكويدًا..

القائمة الرئيسية MainMenu

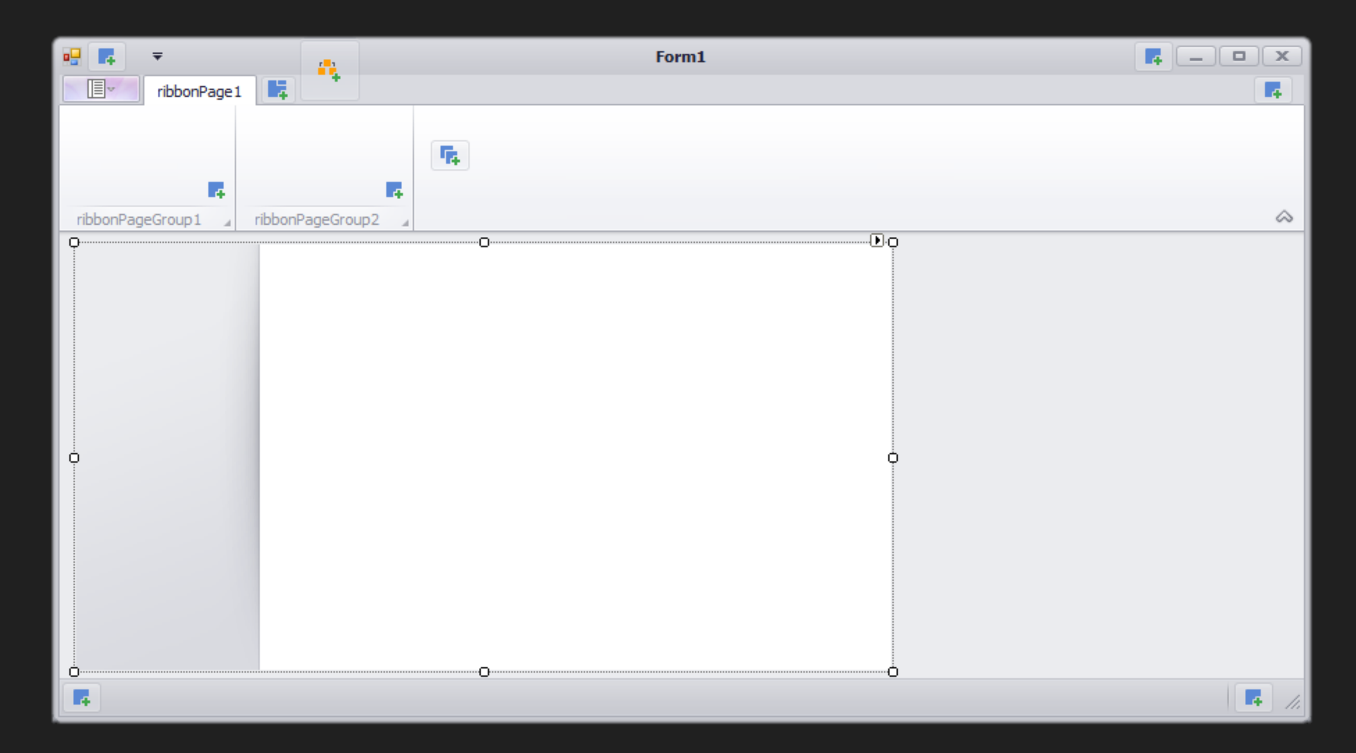
كثيرة هي التطبيقات التي تحوي قوائم رئيسية تحوي العمليات العامة والرئيسية في هذه التطبيقات، وتسمى غالبًا بـ "ملف/File"، وقد تكون على شكل قائمة عادية أو ما يسمى بعرض ما وراء الكواليس Backstage View، وأشهر تطبيق يستخدم هذا النوع من القوائم: تطبيقات الأوفيس. بحيث عندما تنقر على زر "ملف/File" أو ما ينوب عنه (أو ما يمثل القائمة العامة في التطبيق) ستظهر نافذة تملأ التطبيق بالكامل.

في الجزء الأيسر من الشريط Ribbon (أو الأيمن منه إذا حولته للاتجاه العربي) انقر بالزر الأيمن على الزر الموجود في أقصى يسار النافذة (أو يمينها)، ثم انقر على الأمر الثاني:

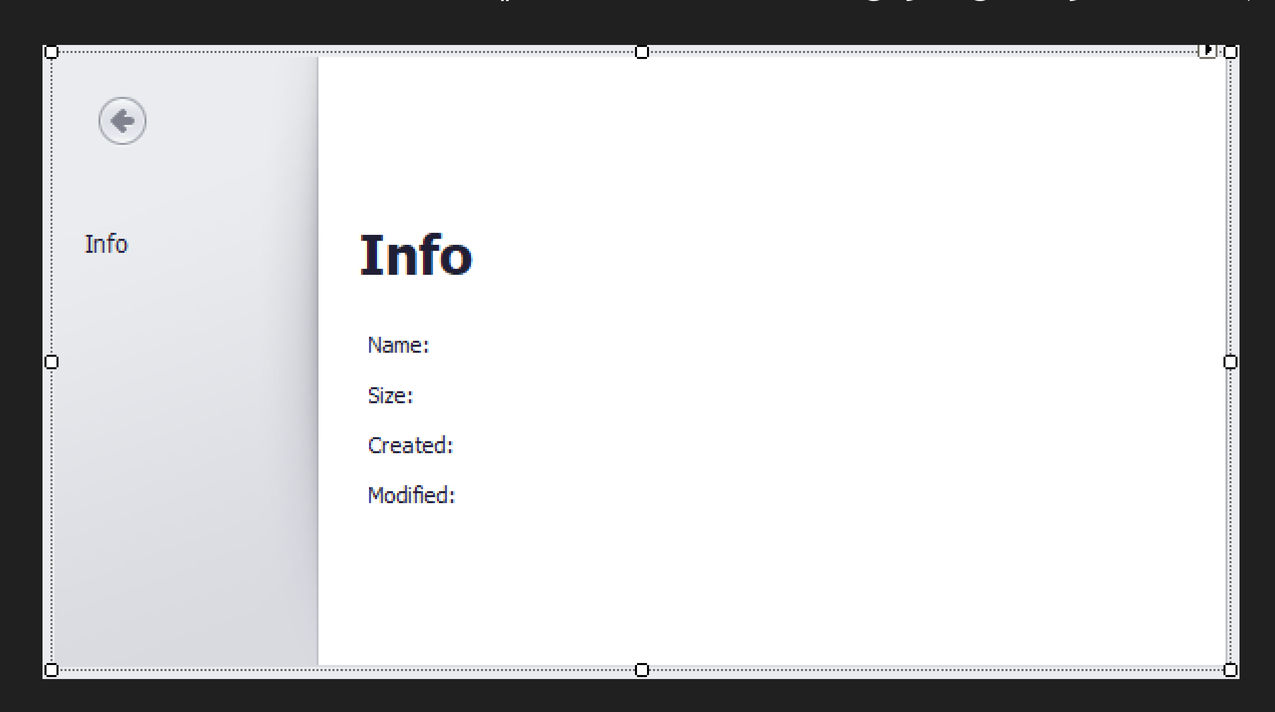


كما يمكنك إضافتها من صندوق الأدوات بإضافة الأداة BackstageViewControl، ثم النقر على اختصار الأوامر المميزة Smart Tag الخاص بالشريط Ribbon، ثم ضمن الفقرة Application Menu وأمام الخاصية Choose اختر اسم الأداة التي أضفتها.

غير حجم النافذة، استعدادًا لضبط خصائص أداة عرض قائمة ما وراء الكواليس:



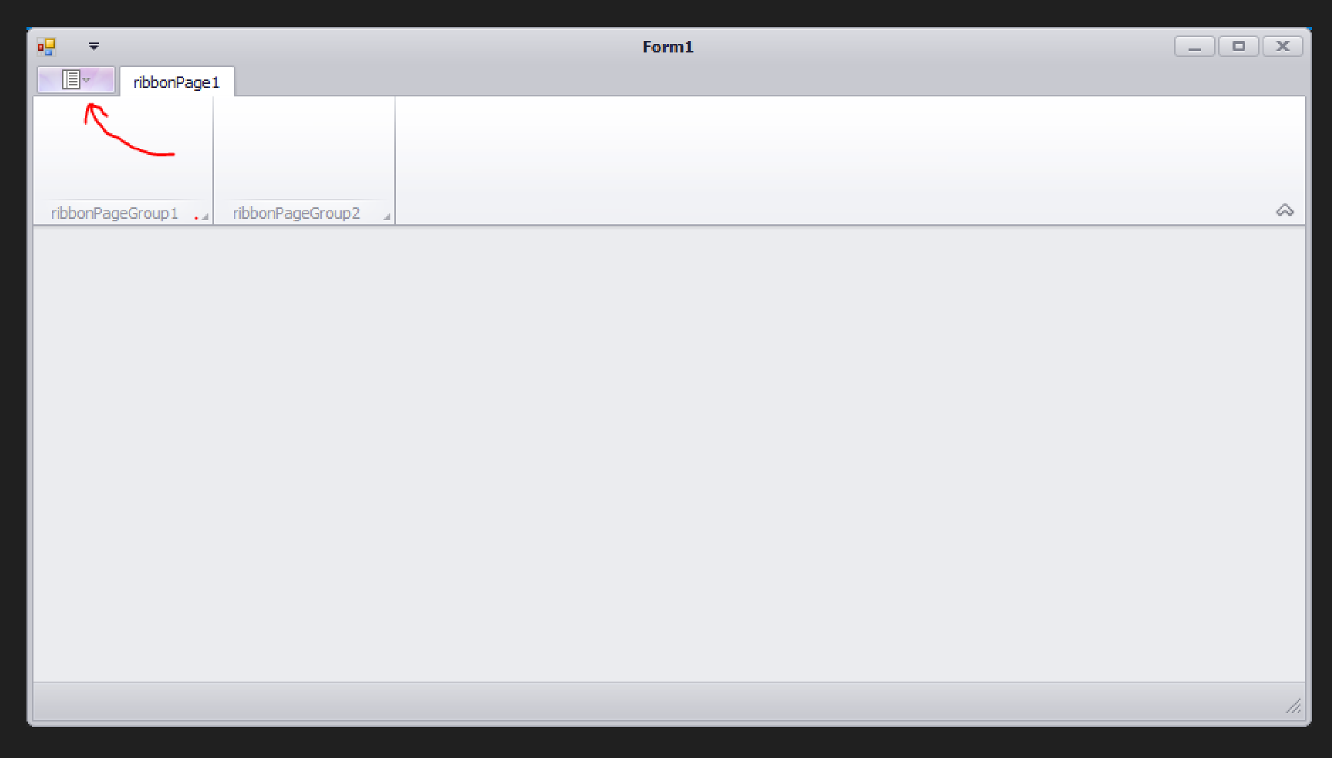
اضف أداة BackstageViewTabItem، لإضافة تبويبة فرعية ضمن أداة BackstageView، ثم أضف مجموعة من اللوائح LabelControls كما يلي:



القيمة	الخاصية	الأداة
Info	Caption	bsvtInfo
-	-	bsvcInfo
20.25	Font.Size	lblInfo
20, 20	Location	
Info	Text	
25, 75	Location	lblInfoName
25, 100	Location	lblInfoSize
25, 125	Location	lblInfoCreated
25, 150	Location	lblInfoModified

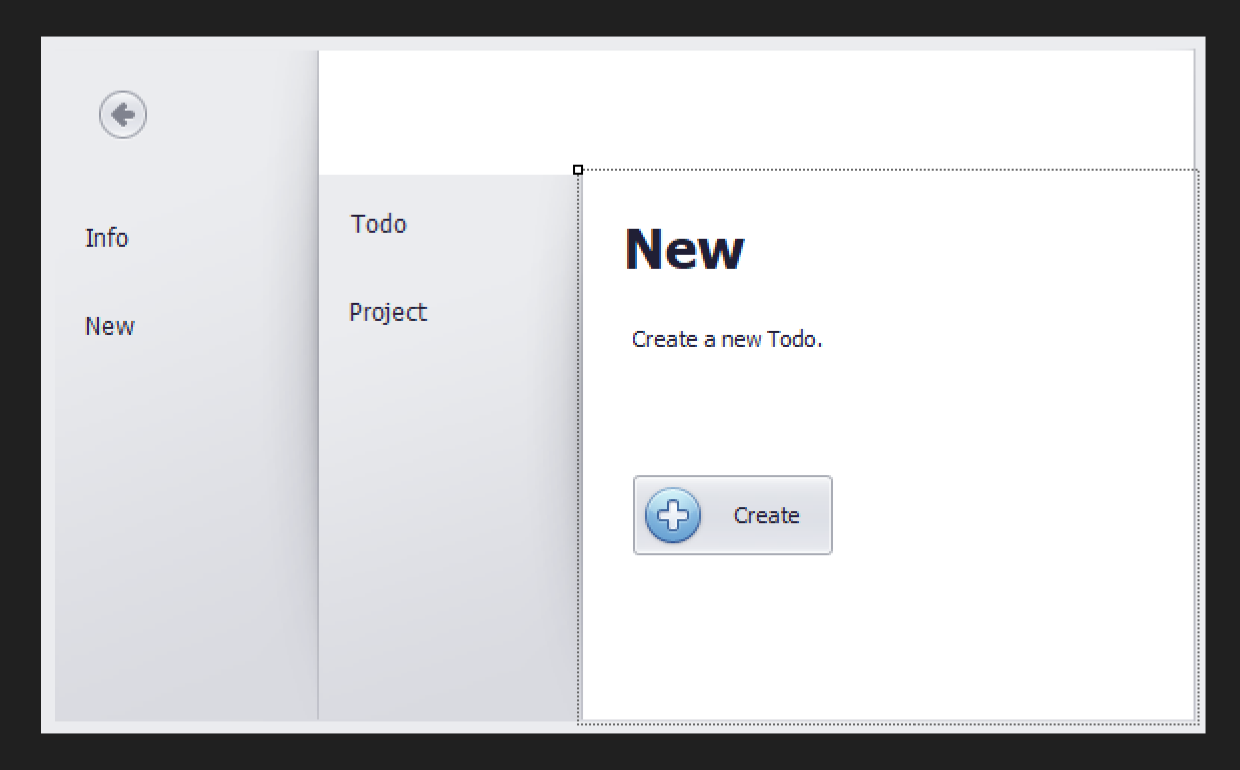
حيث bsvtInfo هي أداة من النوع BackstageViewTabItem وهي تعطيك إمكانية فتح تبويبة عند النقر عليها، أما bsvcInfo فهي أداة من النوع BackstageViewClientControl وهي تمثل التبويبة التي ستظهر عند النقر على الأداة الأولى.

جرب شغل التطبيق، وانقر على زر القائمة الرئيسية:





عد إلى وضع التصميم وانقر بالزر الأيمن على المنطقة الرمادية من أداة BackstageViewControl، أو انقر على الأوامر المميزة Smart Tag للأداة bsvtInfo، ثم اختر الأمر Add Tab، وغير اسمها البرمجي لـ bsvtNew، والاسم الظاهر عليها لـ New. ستحصل مباشرة على أداة من النوع BackstageViewClientControl كتبوبية ستظهر عند النقر على الأداة bsvtNew، انقر على اختصار الأوامر المميزة فيها ثم اختر الأمر إضافة قائمة فرعية Add Child BackstageView. ثم صممها بالشكل التالي:



في البداية أضف أداة BackstageViewTabItem، وفيها ثلاث أدوات هي أداتين من النوع LabelControl وأداة من النوع SimpleButton كما هو موضح بالجدول التالي:

الأداة	الخاصية	القيمة
bsvtNewTodo	Caption	Todo
bsvcNewTodo	-	-
lblNewTodo	Font.Size	20.25
	Location	20, 20
	Text	New

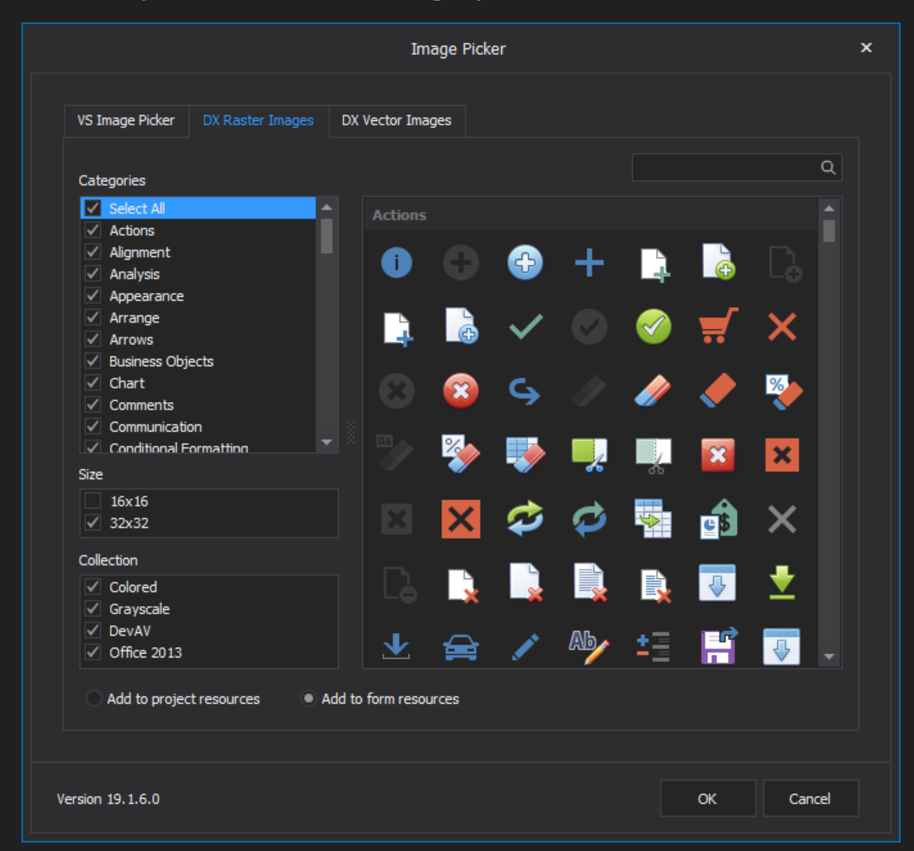
lblNewTodoDesc	Location	25, 75
	Text	Create a new Todo.
btnNewTodo	Image	
	Location	25, 150
	Size	100, 40
	Text	Create

ثم أضف أداة BackstageViewTabItem وفيها نفس الأدوات السابقة، وذلك كما يلي:

الأداة	الخاصية	القيمة
bsvtNewProject	Caption	Project
bsvcNewProject	-	-
lblNewProject	Font.Size	20.25
	Location	20, 20
	Text	New
lblNewProjectDesc	Location	25, 75
	Text	Create a new Project.
btnNewProject	Image	
	Location	25, 150
	Size	100, 40
	Text	Create



يوفر لك الديف إكسبرس مجموعة من الصور الجاهزة لتخدم عددًا كبيرًا من وظائف تطبيقاتك، يمكنك استخدامها لإغناء تطبيقك بالأيقونات.

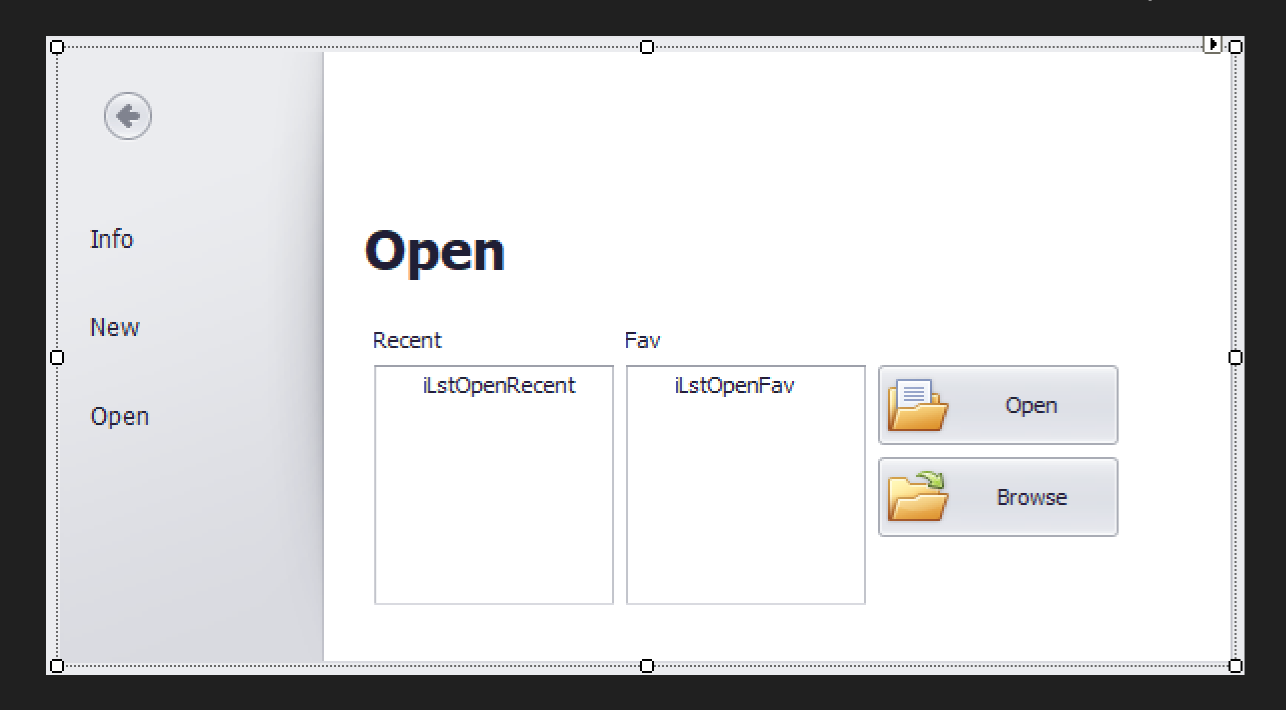




حاول الحفاظ على نسق واحد في تطبيقك، لتجعله سهل الاستخدام. لاحظ أداة LabelControl الأولى ضمن كل تبويبة كيف ضبطنا موقعها بحيث لا تختلف عن التبويبات الأخرى. لاحظ ذلك أيضًا بالنسبة للأدوات الأخرى.

أضف تبويبة لفتح الملفات، والتي يمكن من خلالها استعراض الملفات المفتوحة مؤخرًا أو الملفات المفضلة أو استعراض الكمبيوتر بحثًا عن ملفات.

ضع ضمن التبويبة مجموعة من اللوائح كما هو موضح ضمن التصميم، وأداتين من نوع ImageListBoxControl وأداتين من نوع SimpleButton. وأضف أداة ImageCollection لتحتوي الصور التي ستعرضها صناديق لوائح الصور.



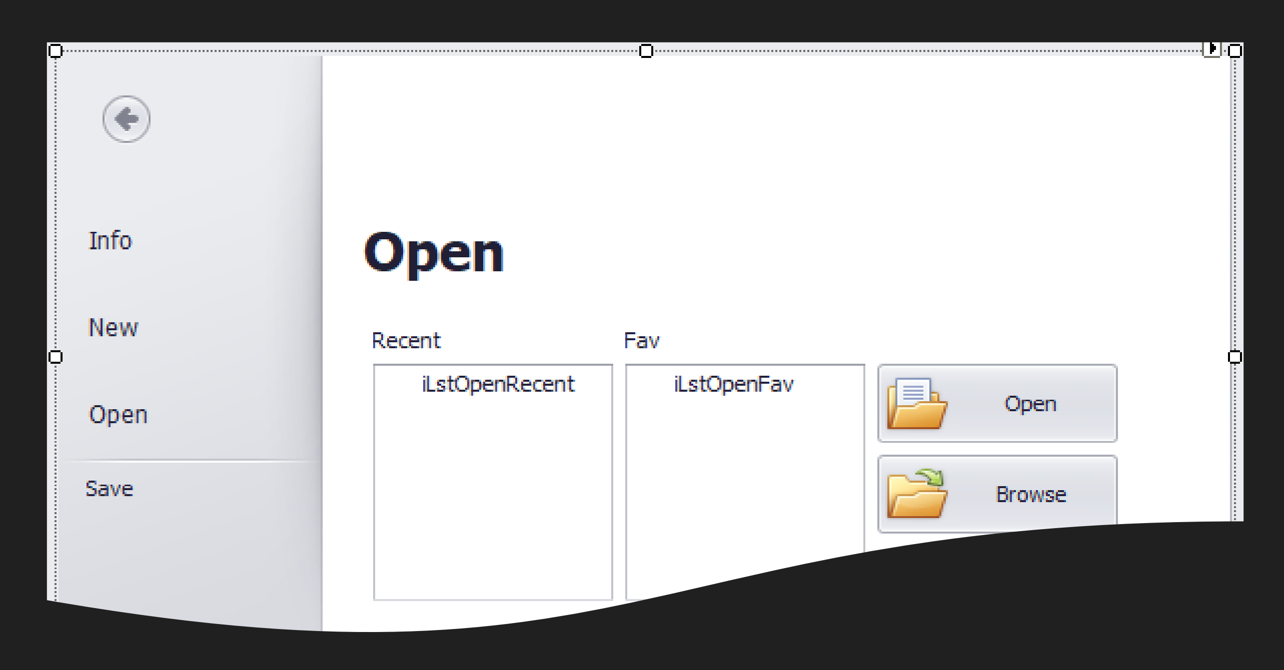
غير اسم الأداة التي ستحوي الصور إلى icOpen (imageCollectionOpen) وأضف إليها صورتين بحجم 16x16 تعبران عن المهام (ما يدل على ملف إفرادي) والمشاريع (ما يدل على ملف يمثل مجموعة ملفات).

استعن بالجدول التالي لضبط خصائص الأدوات:

الأداة	الخاصية	القيمة
bsvtOpen	Caption	Open
bsvcOpen	-	-
lblOpen	Font.Size	20.25
	Location	20, 20
	Text	Open
btnOpenRecent	Image	
	Location	277, 94
	Size	120, 40
	Text	Open

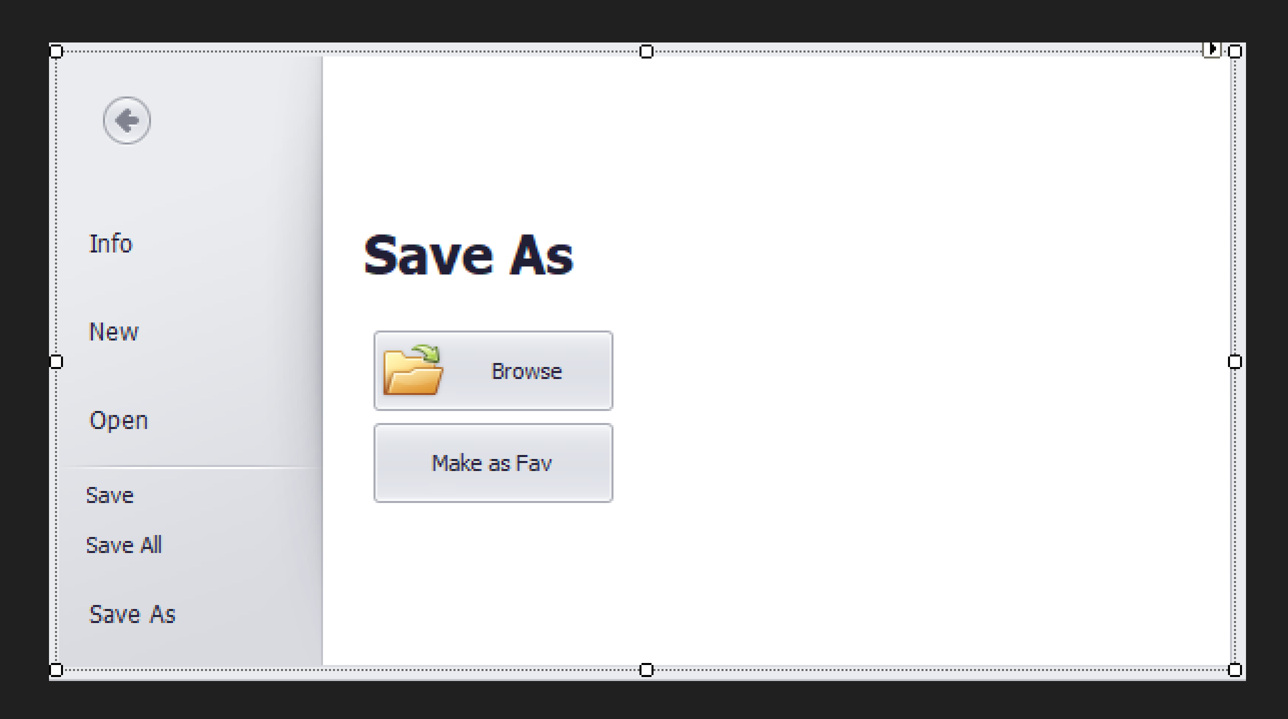
btnOpenBrowse	Image	
	Location	277, 140
	Size	120, 40
	Text	Browse
icOpen	Images	
	ابحث عن الكلمة file	
iLstOpenRecent	ImageList	icOpen
	Location	25, 94
	Size	120, 120
iLstOpenRecent	ImageList	icOpen
	Location	151, 94
	Size	120, 120

أضف فاصلًا Separator، ثم أمرًا Command لحفظ الملف الحالي:



لاحظ أن الحجم الافتراضي للأوامر Commands أصغر من ذلك الذي تتمتع به التبويبات. غير الاسم البرمجي لهذا الأمر إلى bsvbSave والعنوان الظاهر عليه لـ Save.

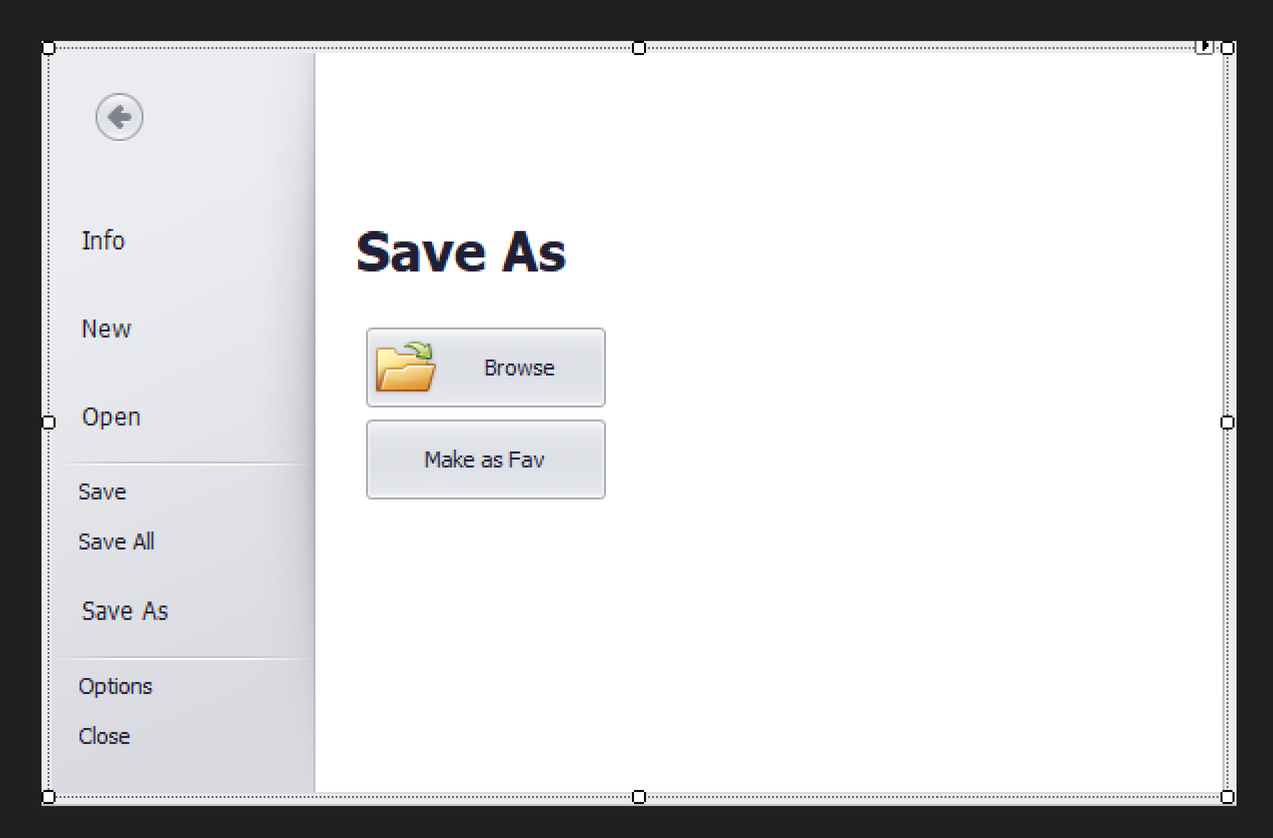
بالمثل، أضف أمرًا باسم bsvbSaveAll وغير العنوان الظاهر عليه لـ Save All. ثم أضف تبويبة باسم bsvtSaveAs وصممها كما يلي:



الأداة	الخاصية	القيمة
bsvtSaveAs	Caption	Save As
bsvcSaveAs	-	-
lblSaveAs	Font.Size	20.25
	Location	20, 20
	Text	Save As
btnSaveAs	Image	
	Location	25, 75
	Size	120, 40
	Text	Browse

<table dir="rtl">
<tr><td rowspan="4">cbtnFavOrNot</td><td>Location</td><td>25, 121</td></tr>
<tr><td>Size</td><td>120, 40</td></tr>
<tr><td>Text</td><td>Make as Fav</td></tr>
<tr><td colspan="2">هذه الأداة من النوع CheckButton</td></tr>
<tr><td rowspan="3">icFavOrNot</td><td rowspan="2">Images</td><td></td></tr>
<tr><td></td></tr>
<tr><td colspan="2">هذه الأداة ستحوي صورًا تُعرض على أداة
cbtnFaveOrNot
هذه الصور بحجم 32x32</td></tr>
</table>

كبّر الأداة التي ستحوي القائمة لتتسع لأوامر أكثر، ثم أضف فاصلًا Separator ثم أمرين أحدهما باسم bsvbClose لإغلاق التطبيق والآخر باسم bsvbOptions لفتح نافذة إعدادات البرنامج:



شغل التطبيق وتنقل بين القوائم، ولاحظ مجريات الأمور، وحاول تطويرها كما في البرامج المشهورة لتحوي أكثر الأوامر شيوعًا والتي ستختصر على المستخدمين الوصول لوظائف تطبيقك العامة.

الآن، قم بإخفاء القائمة في مكان ما حتى لا تعيق عملنا أثناء تصميم الأجزاء الأخرى من البرنامج (حركها إلى أسفل النافذة مثلًا)، من المفترض أن هناك خاصية تخفي الأداة أثناء التصميم لكني لم أجدها (في نسختي على أقل تقدير).

القائمة الرئيسية MainMenu، بعمق

القائمة الرئيسية هي قائمة تظهر عند النقر على زر التطبيق Application Button أو ما يمكن أن نسميه الزر الرئيسي في التطبيق، حيث إننا في الفقرة السابقة لم نكن نضبط إلا القائمة التي ستظهر على زر التطبيق. يمكنك ضبط الكثير من الخصائص التي لها دور كبير في ضبط الزر الرئيسي في التطبيق، وتشترك هذه الخصائص جميعها بوجود الكلمتين Application وButton في أسماءها.

يمكنك إظهار أو إخفاء الزر الرئيسي في التطبيق من خلال الخاصية ShowApplicationMenu، وللخاصية فائدة كبيرة عند التعامل مع التطبيقات متعددة المستندات، بحيث تكون النوافذ الأبناء – أو بعضها – والنافذة الأم مبنية على أساس الشريط، أي أنها Ribbon MDI Applications. كما يمكنك الاستفادة عموما من هذه الخاصية عندما لا ترغب بوجود الزر الرئيسي حتى لو كان تطبيقك مبني على نافذة واحدة.

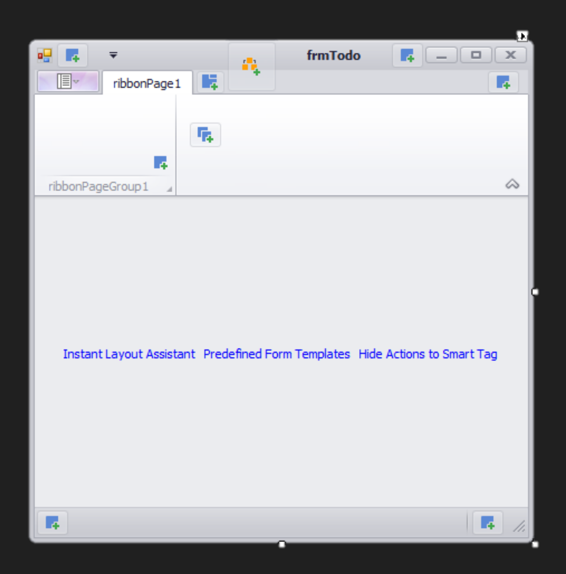
يمكنك ضبط القائمة التي ستظهر عند النقر على زر التطبيق – وهي ما شرحناه في الفقرة السابقة – من خلال الخاصية ApplicationButtonDropDownControl، وتعيين أيقونة الزر الرئيسي من خلال الخاصية ApplicationButtonImageOptions. يمكنك إضافة تلميح يظهر عند مرور المؤشر على الزر الرئيسي من خلال الخاصية ApplicationButtonSuperTip. كما يمكنك عرض نص عوضًا عن صورة كما في تطبيقات الأوفيس (تجد زر التطبيق باسم File)، وذلك من خلال الخاصية ApplicationButtonText.

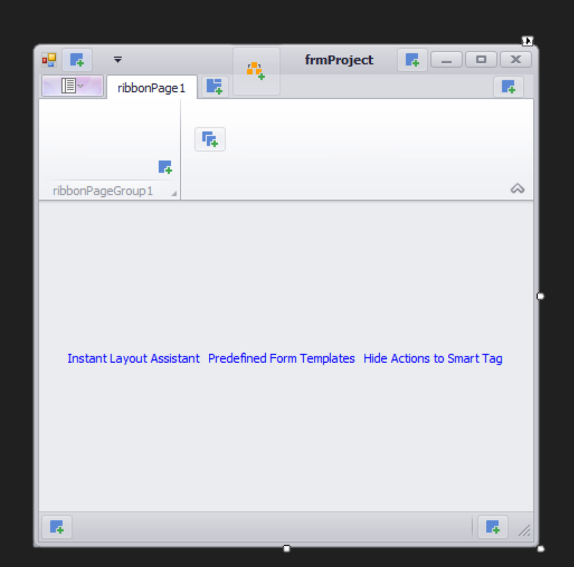
النوافذ متعددة المستندات MDI Forms

تعطيك بعض البرامج إمكانية التعامل مع أكثر من مستند في الوقت ذاته (مثل الفيجوال ستديو)، بينما لا تدعم غيرها ذلك (مثل ميكروسوفت وورد). سنضيف إمكانية التعامل مع أكثر من مستند في هذا المشروع لإغنائه وإثراء أفكاره.

اضبط الخاصية IsMdiContainer على القيمة True في النافذة الرئيسية في المشروع – كان اسمها Form1 – وأضف نافذتين من نوع الشرائط، الأولى لعرض ملف المهمة (والذي يحوي تفاصيل هذه المهمة بالكامل) والثانية لعرض المشاريع (مجموعات المهام).

من المفترض أنه لديك الآن النافذتين:





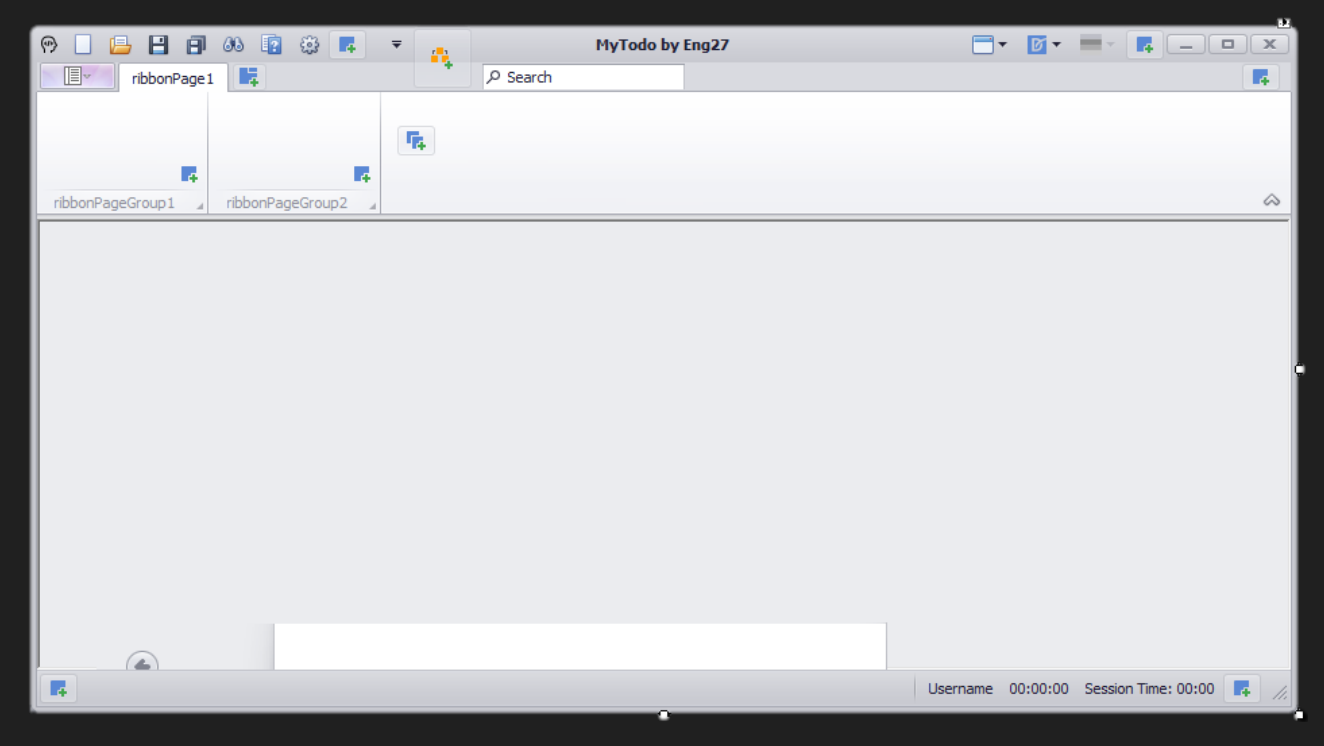
تهيئة النموذج الأم في المشروع

النموذج الرئيسي في التطبيق يجب أن يحوي مجموعة من الوظائف التي تجعل استخدام التطبيق أسهل، بما في ذلك تلك الوظائف التي ستؤثر على النماذج الأبناء أو الوظائف التي ستؤثر على المشروع عمومًا (مثل المظهر والإعدادات والمساعدة وغيرها). وما سنقوم به في هذه الفقرة هو تهيئة النموذج الأم وإضافة الأدوات والعناصر المناسبة حتى يصبح المشروع مفيدًا وذي معنى.



العناصر Items هي أدوات تضاف إلى أدوات Controls ديف إكسبريس. وتسمى أيضًا روابط Links. فمثلًا RibbonControl هي أداة بينما BarButtonItem هي عنصر.

اضبط وأضف مجموعة من الأدوات كما يلي (الأدوات في الجدول غير مرتبة أبجديًّا):

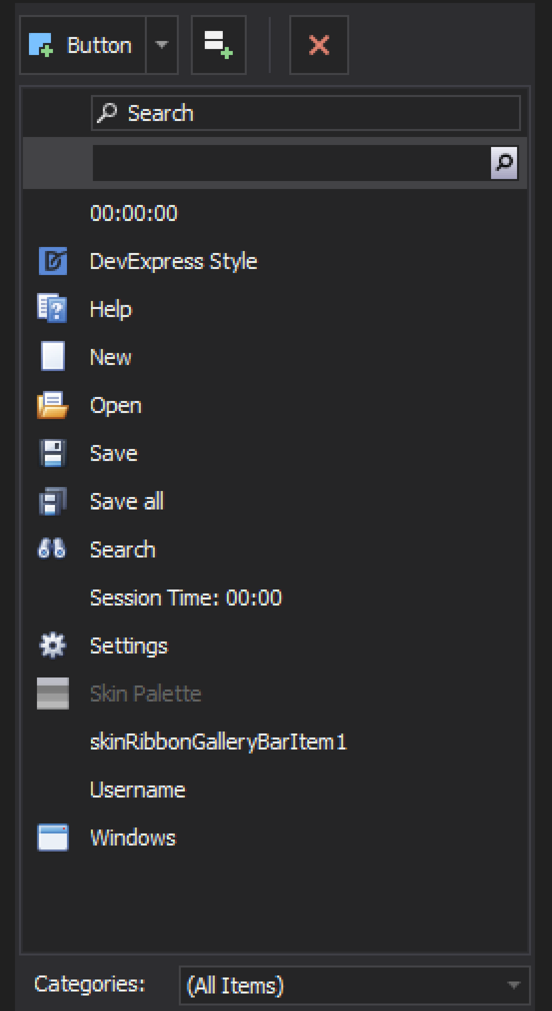


الأداة	الخاصية	القيمة
Form1	Text	MyTodo by Eng27
	Icon	
ribbonControl1	ShowSearchItem	True
الأدوات التالية موجودة في شريط العنوان (يسمى Caption Bar أو Title Bar)		
barMdiChildrenListItem1	Caption	Windows
	Image	

-	-	skinDropDownButtonItem1
-	-	skinPaletteDropDownButtonItem1
الأدوات التالية موجودة في شريط الوصول السريع (يسمى QAT)		
New	Caption	bbtnNew
	Image	
Open	Caption	bbtnOpen
	Image	
Save	Caption	bbtnSave
	Image	
Save all	Caption	bbtnSaveAll
	Image	
Search	Caption	bbtnSearch
	Image	
Help	Caption	bbtnHelp
	Image	
Settings	Caption	bbtnSettings
	Image	
الأدوات التالية موجودة في شريط الحالة (يسمى Status Bar) وهي جميعها من نوع BarStaticItem Username 00:00:00 Session Time: 00:00		
Username	Caption	bsUsername
00:00:00	Caption	bsTime

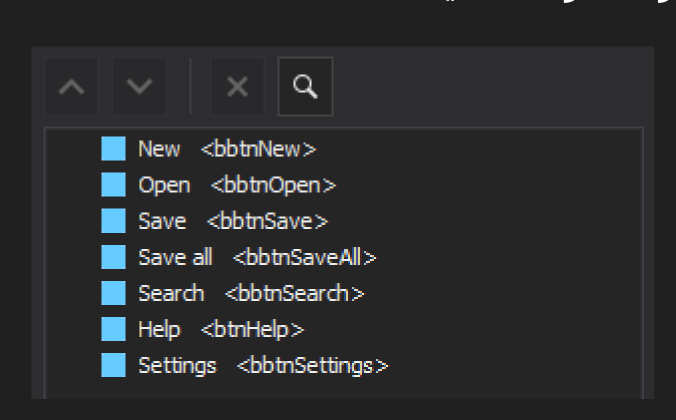
bsSessionTime	Caption	Session Time: 00:00

بعض الأدوات سنناقشها لاحقًا، ما يهمني من هذه الفقرة هو كيفية إضافة وتنسيق الأدوات للأشرطة الرئيسية، بغض النظر عن معنى هذه الأدوات.

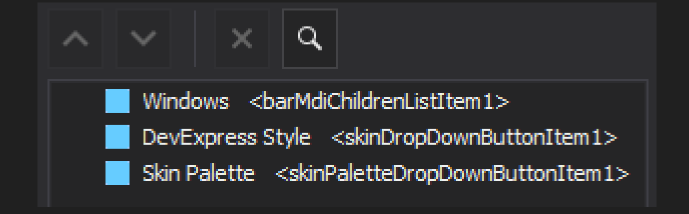


الآن، انقر بالزر الأيمن على مكان خال من الشريط، ثم Run Designer، ستلاحظ وجود جميع الأدوات التي أضفتها للنافذة ضمن القسم الثالث من المصمم، كما هو واضح في الصورة المجاورة.

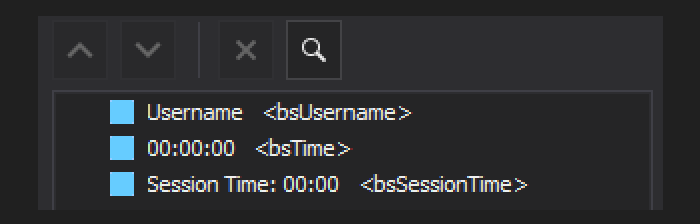
إذا انتقلت لشريط الوصول السريع QA Toolbar ضمن القسم الأول من المصمم فإنك ستلاحظ وجود الأدوات التالية:



أما إذا انتقلت لشريط العنوان Caption Bar:



وضمن شريط الحالة Status Bar ستجد ما يلي:



وبقية الأقسام خالية لا أدوات فيها.

الجميل في المصمم أنه يعطيك إمكانية استخدام نفس الأداة ضمن أكثر من مكان من البرنامج، فبعد إنشاء الأدوات سيتم إضافتها للقسم الثالث من المصمم، ومنه يمكنك سحبها لأي مكان في البرنامج بحيث إذا عدلتها ضمن مكان ما ستتعدل في الأماكن الأخرى، وهذا ما سنراه لاحقًا.

تهيئة النموذج الأم في المشروع، بعمق

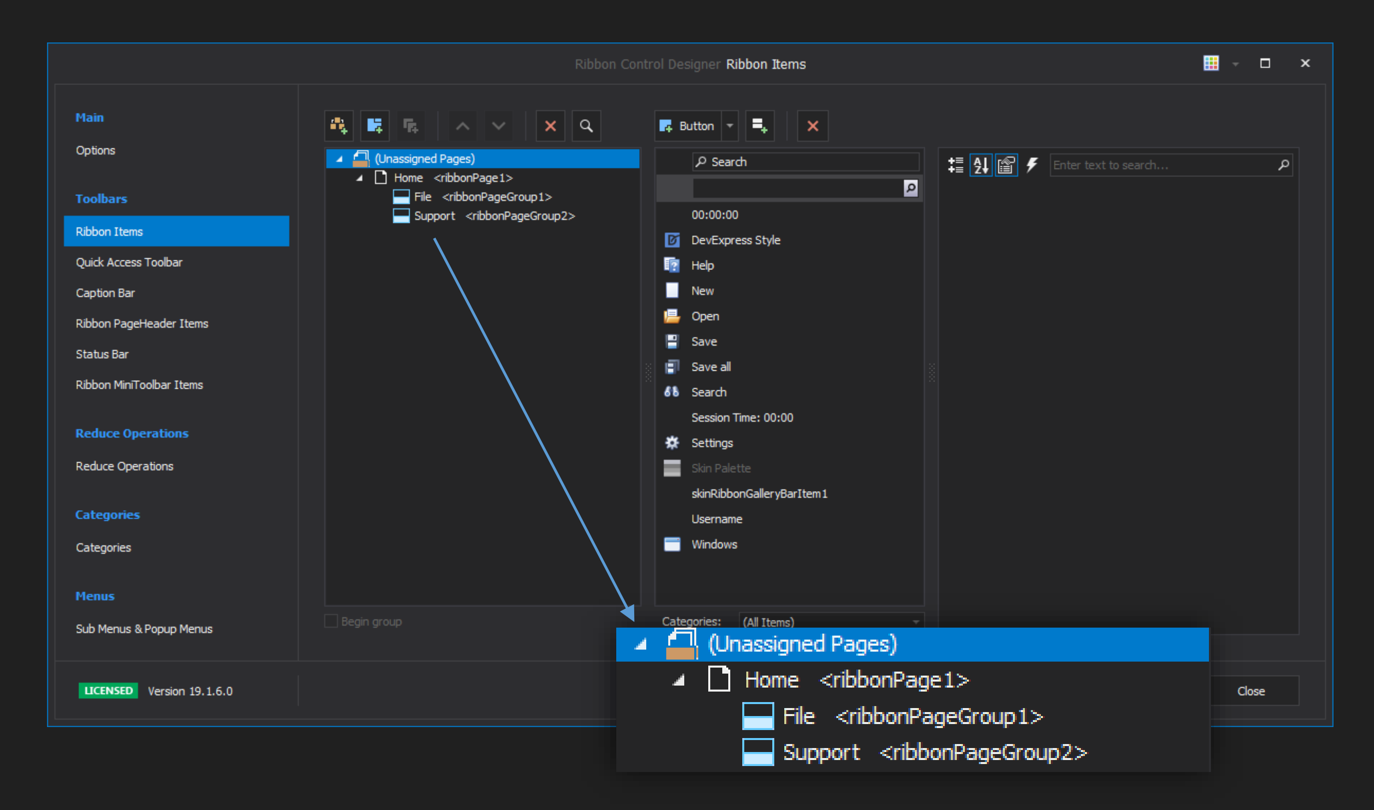
وضعنا في الفقرة السابقة بعض أساسات المشروع والتي ستلعب دورًا كبيرًا في تسهيل استخدام التطبيق على المستخدم، ألا وهي الاختصارات وبعض الأوامر الرئيسية. وفي هذه الفقرة سنضع الخطوط العريضة التي ستشكل ملامح التطبيق ومزاياه ووظائفه، والتي تتمثل بصفحات الشريط Ribbon Pages.

الصفحة الأولى – المضافة أصلًا – هي الصفحة الرئيسية من التطبيق، والتي غالبا تحمل الاسم "Home"، كما يمكنها أن تحمل أسماءً أخرى (خصوصًا تلك البرامج التي لا تملك صفحة رئيسية). في مشروعنا، الصفحة الرئيسية معنية بالأوامر العامة في التطبيق، مثل إنشاء وفتح الملفات، الإعدادات والحصول على المساعدة، بالإضافة لمجموعة من الأوامر المتعلقة بالملفات التي سيتعامل معها التطبيق (التطبيق سيتعامل مع نوعين من الملفات، الأول ملفات تمثل المهام، الثاني ملفات تمثل المشاريع، ولكل من النوعين السابقين أوامر خاصة به).

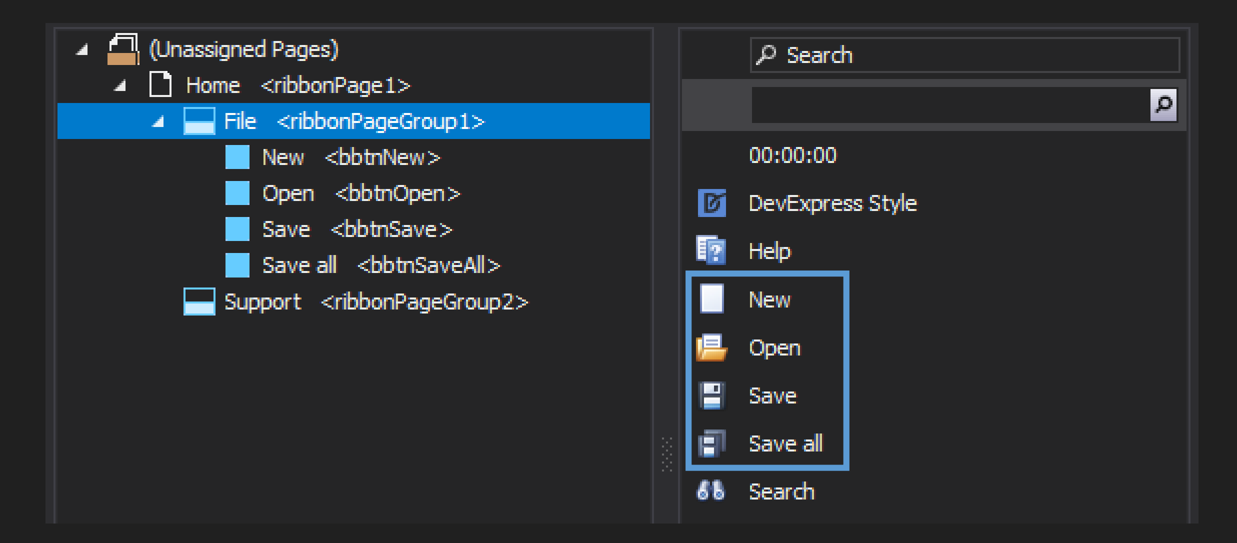
حاول دائمًا جعل غالب عمل المستخدم في الصفحة الرئيسية أو العامة، واجعل الأوامر المتعلقة بالتفاصيل الثانوية أو الاختصاصات الدقيقة ضمن التطبيق موجودة في الصفحات الأخرى.

ستحوي الصفحة الأولى – على سبيل المثال – المجموعتين: ملف File، والدعم Support (والذي سيحوي الإعدادات والمساعدة).

افتح المصمم وغير تسمية الصفحة والمجموعتين كما يلي:



اسحب العناصر: جديد New، وفتح Open، وحفظ Save، وحفظ الكل Save all إلى المجموعة الأولى ورتبهم كما يلي:



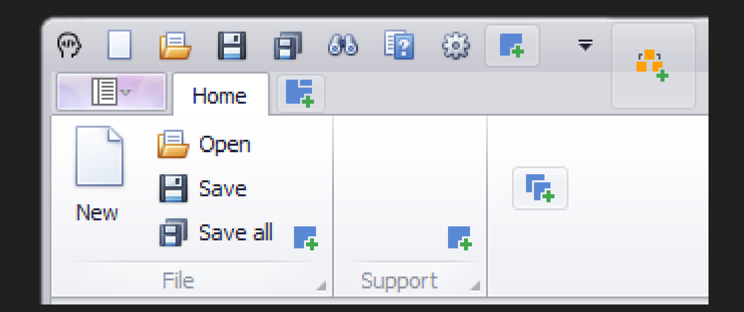
أغلق المصمم وتأمل النتيجة:



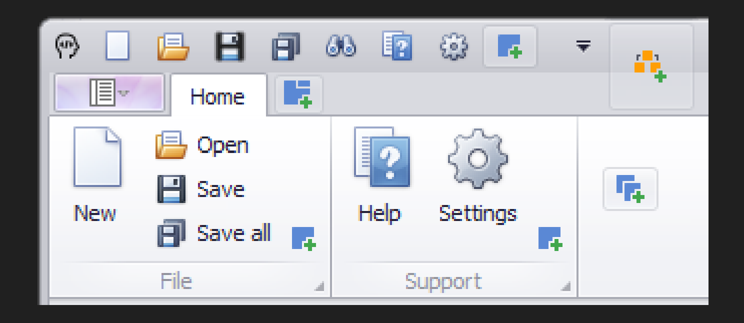


يمكنك نسخ العناصر واستخدامها في أكثر من مكان من البرنامج دون الاعتماد على المصمم وذلك بسحبها مع الاستمرار بالضغط على زر Ctrl.

غير الخاصة RibbonStyle لكل العناصر عدا الأول، واجعلها SmallWithText لتصبح المجموعة بالشكل:



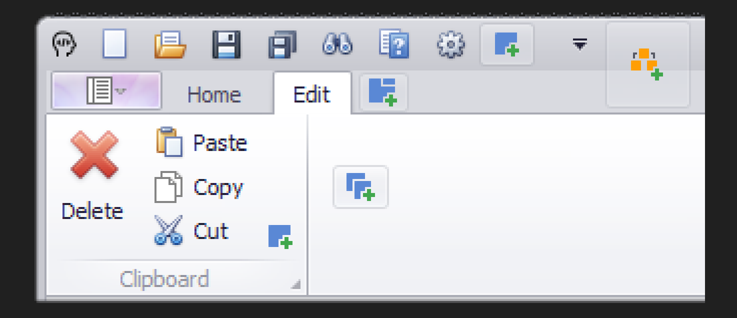
أفضل، صحيح؟؟ كرر الخطوة من أجل مجموعة الدعم، جرب سحبها مع نسخها دون الاعتماد على المصمم:



غير الأسماء البرمجية للصفحات والمجموعات والعناصر المستخدمة، اجعل أسماءها تبدأ بنوعها، الصفحات تبدأ بـ rb والمجموعات تبدأ بـ rpg. فمثلًا الصفحة الأولى سمّها rbHome. صحيح أني غالبًا لا أغير أسماء الأدوات في الأكواد التي أضعها في كتبي – مثل جعل أسماء الأدوات تصف عمل هذه الأدوات وتبدأ بـ txt وbtn وغيرها – خصوصًا عندما يحوي المشروع أدوات معدودة، ولكني مع ذلك لا أحبّذ هذا الأسلوب. عوّد نفسك على الاختصارات، خصوصًا المشهورة منها بين أوساط المبرمجين.

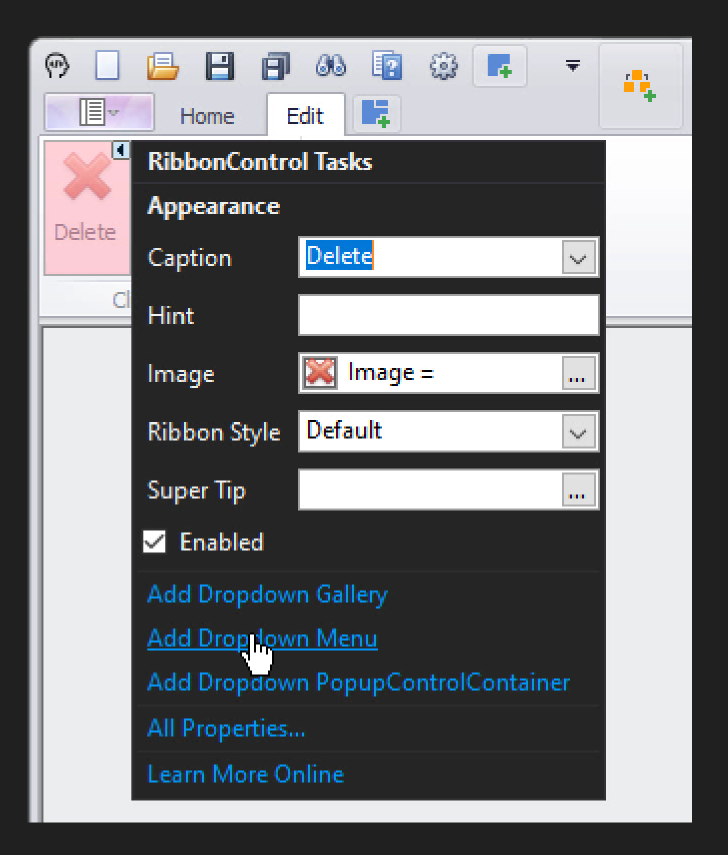
الصفحة الثانية هي صفحة تعديل الملفات Edit، ومن خلالها سيتمكن المستخدم من التعامل مع الملفات المختلفة وتعديلها. سنضع في هذه الصفحة المجموعات: الحافظة Clipboard، والأيقونة Icon، والبحث والاستبدال Find and Replace.

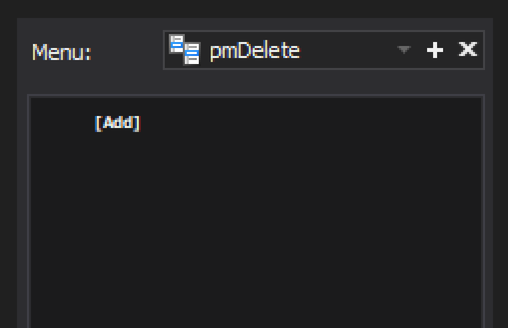
المجموعة الأولى فيها أربعة أوامر:



على اعتبار أن العناصر الأربعة الموضحة بالصورة السابقة لم نضفها سابقًا فيجب عليك إنشاءها، سمها برمجيا واضبط صورها ومواقعها بما هو مناسب لتحصل على ما هو قريب مما في الصورة السابقة.

ولإغناء المشروع، انقر على الأوامر المميزة Smart Tag الخاصة بالعنصر bbtnPaste واختر Add Dropdown Menu:

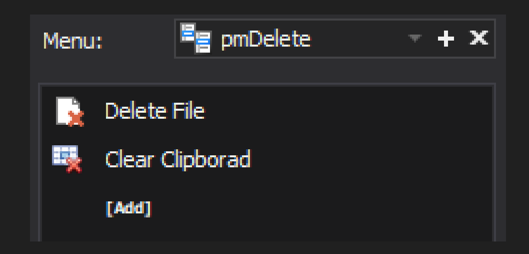




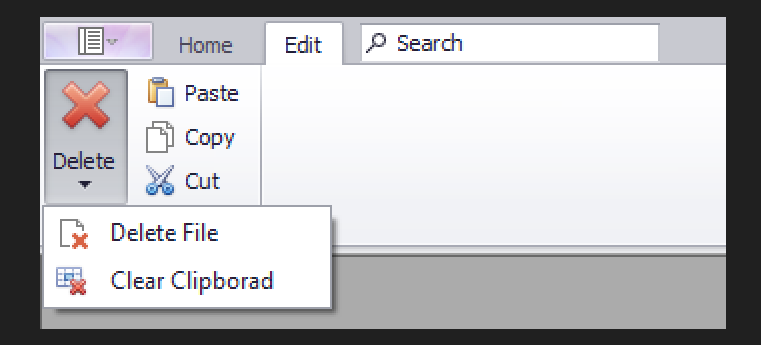
ستحصل على أداة من النوع PopupMenu، غيّر اسمها لـ pmDelete وانقر على الـ Smart Tag الخاص بها ثم Run Designer لتفتح المصمم، والذي بدوره ينقلك للقسم الذي يتم فيه ضبط القوائم، وبشكل طبيعي ستكون القائمة فارغة على اعتبار أنك أنشأتها حديثًا.

ما نرغب بإضافته للقائمة هو أوامر متفرعة عن الحذف، وهي حذف الحقل المحدد (الخيار الافتراضي)، وحذف الملف، وإفراغ الحافظة Clipboard، كما يمكنك إضافة غيرها. وقد لا تكون هذه الأوامر بهذه الأهمية ضمن التطبيق، كتطبيق حقيقي، لكن الغاية منها ضمن المشروع هو إيضاح كيفية إضافة زر له فروع.

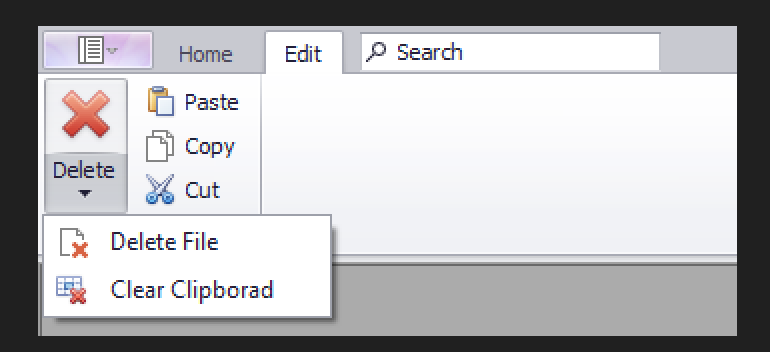
انقر على [Add] واختر Button، وغير اسمه البرمجي لـ bbtnDeleteFile وعنوانه لـ Delete File، وأضف زر آخر وغير اسمه البرمجي لـ bbtnClearClipboard وعنوانه لـ Clear Clipboard، واختر لهما صورًا مناسبة على ذوقك وفق ما يلي:



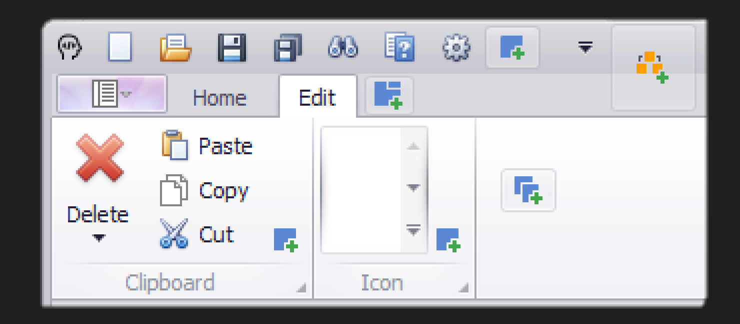
شغل المشروع، ولاحظ:



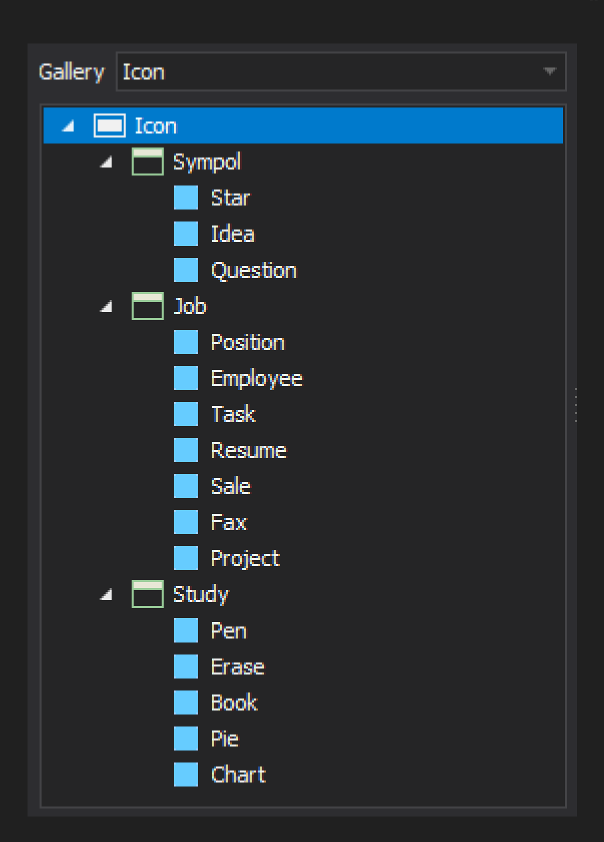
لاحظ أنه في هذه الحالة عند النقر على الزر Delete يتم فتح قائمة فيها مجموعة من الأوامر. وإن كنت ترغب بإنشاء قائمة تُفتح عند النقر على سهم أسفل الزر، مع الإبقاء على هذا الزر كزر له وظيفة خاصة به، فألغ تفعيل الخاصية ActAsDropDown:



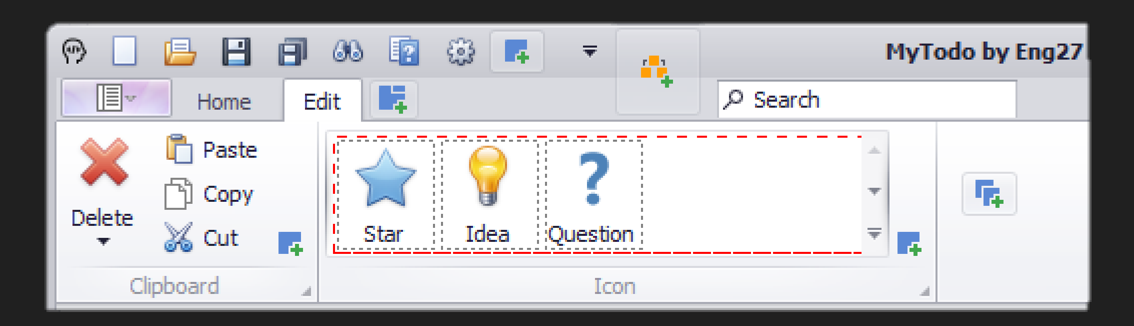
المجموعة الثانية فيها عنصر واحد فقط، وهو من النوع RibbonGalleryBarItem:



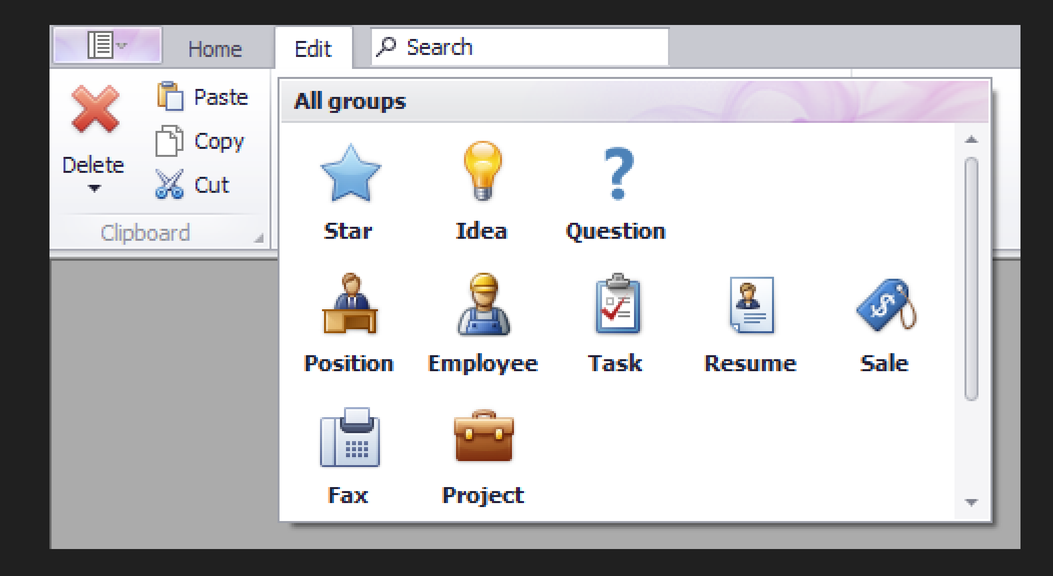
غير اسم الأداة التي تم إضافتها إلى rgbIcon، والخاصية Caption إلى Icon، وضمن الخاصية Gallery غير الخاصية الفرعية ItemCheckMode إلى SingleCheck، والخاصية ImageSize إلى 32, 32، والخاصية ShowItemText إلى True. ثم انقر على الأوامر المميزة لها واختر Run Designer، ثم أضف ثلاثة مجموعات ولكل مجموعة بضعة عناصر كما هو موضح بالشكل (يمكنك البحث عن صور العناصر بالبحث عن العناوين المعروضة عليها في الشكل التالي):



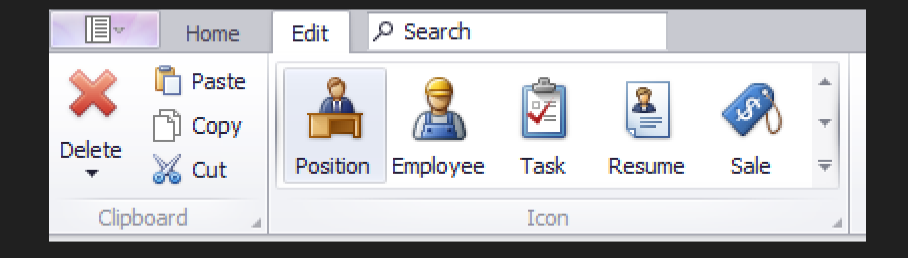
وعندها:



شغل التطبيق:

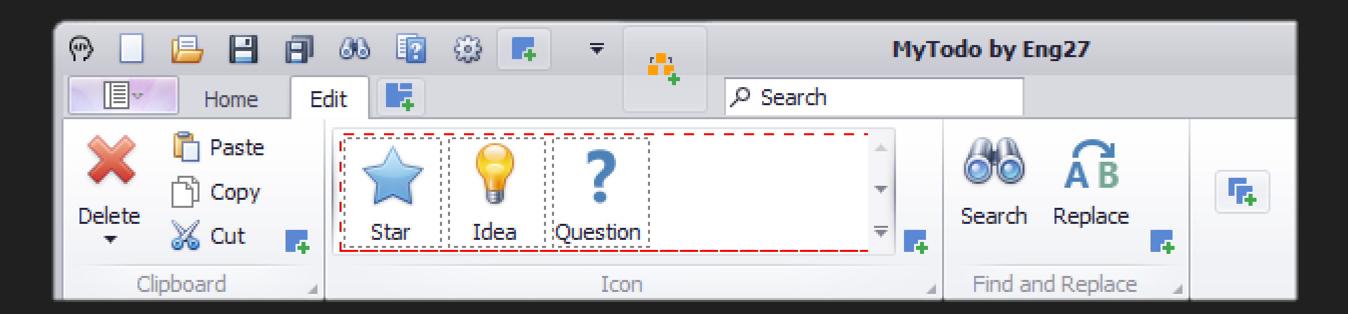


كما يمكن التنقل دون فتح القائمة من خلال الأسهم الموجودة على يمين الأداة:



ولمزيد من التخصيص، هناك الكثير من الخصائص التي تلعب دورًا في شكل وتصرف الأداة RibbonGalleryBarItem، وكلها موجودة ضمن الخاصية Gallery، جرب تغييرها والتعديل عليها ولاحظ النتيجة.

المجموعة الثالثة فيها عنصران، الأول زر البحث والثاني زر الاستبدال. يمكنك إضافة زر البحث بسحبه مع النسخ من شريط الوصول السريع، بينما الزر الآخر يجب إضافته من البداية.



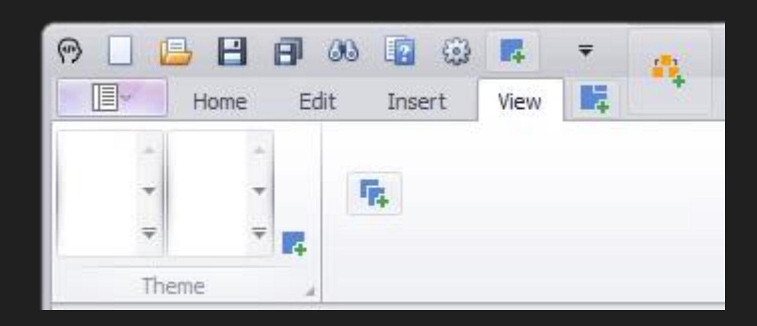


إذا أردت تفعيل تكبير الصورة عند مرور مؤشر الفأرة عليها ImageHovering اضبط الخاصية ImageSize للخاصية Gallery على القيمة 16, 16 واختر الصور بهذا الحجم، ثم أضف صورًا من الحجم 32, 32 للخاصية HoverImage.

الصفحة الثالثة هي صفحة الإضافة Insert، ومن خلالها يمكن للمستخدم إضافة الملفات أو محتواها. الصفحة الثالثة لا تحوي أوامر مبدئيا، على أن نضيف إليها مجموعات معينة بناءً على نوع الملف المفتوح.

الصفحة الرابعة هي صفحة العرض View، ومن خلالها يمكن للمستخدم تغيير خيارات العرض. وفيها مجموعتين، الأولى مجموعة السمة Theme، والثانية مجموعة النوافذ Windows.

أضف للمجموعة الأولى العنصرين Skin Galleryو Skin Palette Gallery:



أما للمجموعة الثانية فأضف عنصرًا من النوع BarMdiChildrenListItem – لتحصل على الأداة barMdiChildrenListItem2 – وغير عنوانه لـ Windows وخاصيته RibbonStyle لـ Large وصورته لما هو موضح بالشكل التالي:

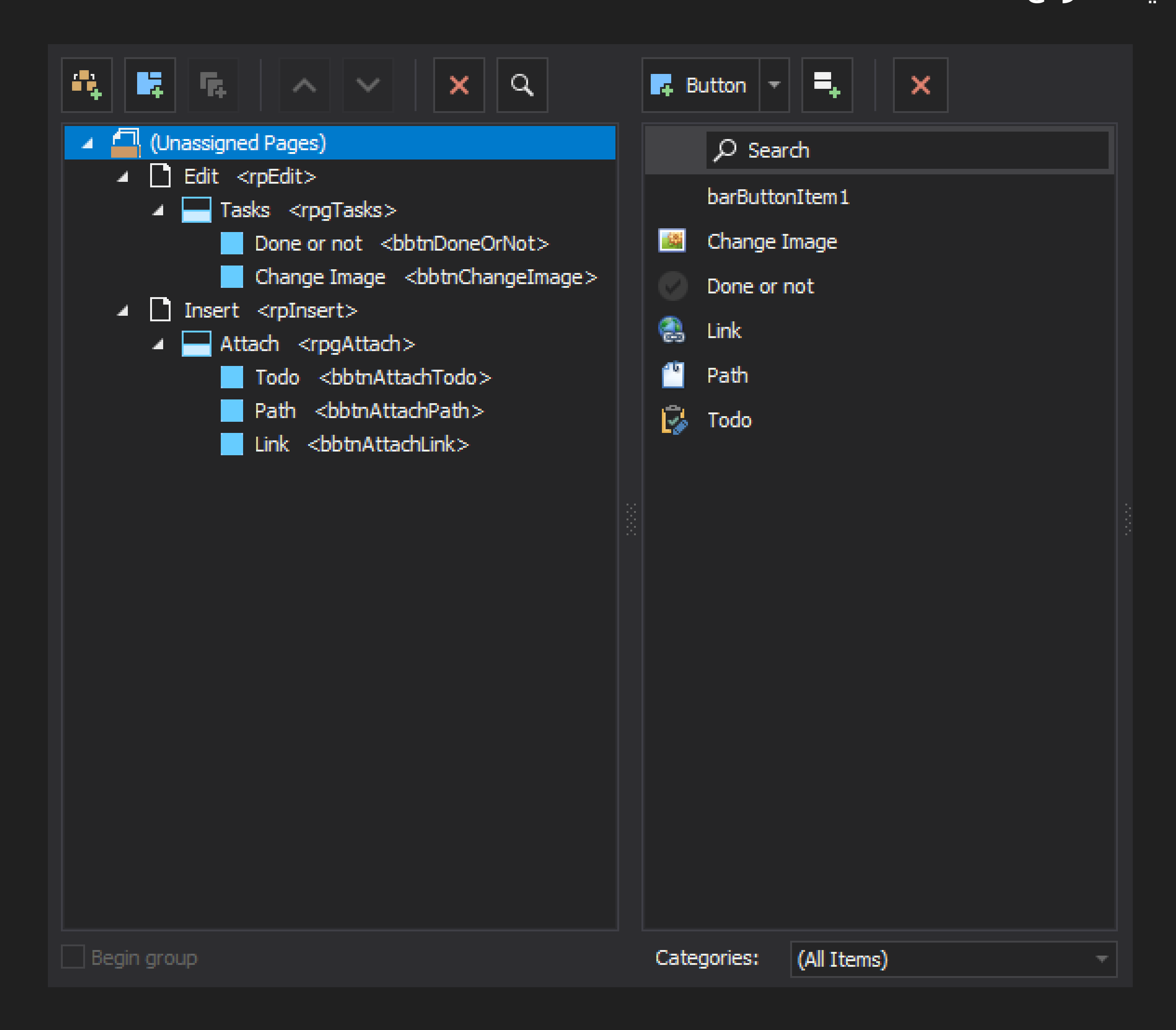


تهيئة النماذج الأبناء

عند التعامل مع النماذج الأبناء من خلال النموذج الأم سيتم إضافة عناصر شريط النموذج الابن إلى عناصر شريط النموذج الأم، لذلك ما سنقوم به في هذه الفقرة هو إنشاء الأوامر المتعلقة بكل نموذج، مع العلم أن الأوامر المشتركة بالنموذجين – نموذج الملفات التي تمثل ملفًّا بذاته Todo file ونموذج الملفات التي تمثل ملفًّا جامعًا Project file – قمنا بإضافتها لشريط النموذج الأم.

وعلى اعتبار أن النماذج الأبناء من النوع RibbonForm فإنها ستحوي زرًّا رئيسيًّا، وهو ما لا نرغب به (على اعتبار وجود زر رئيسي في النموذج الأم)، لذلك سنقوم بدايةً بإلغاء تفعيل الخاصية ShowApplicationButton لشريط كل من النموذجين frmTodo وfrmProject. كما أنه لا حاجة لظهور أيقونة النوافذ الأبناء، لذلك فألغ تفعيل الخاصية ShowIcon.

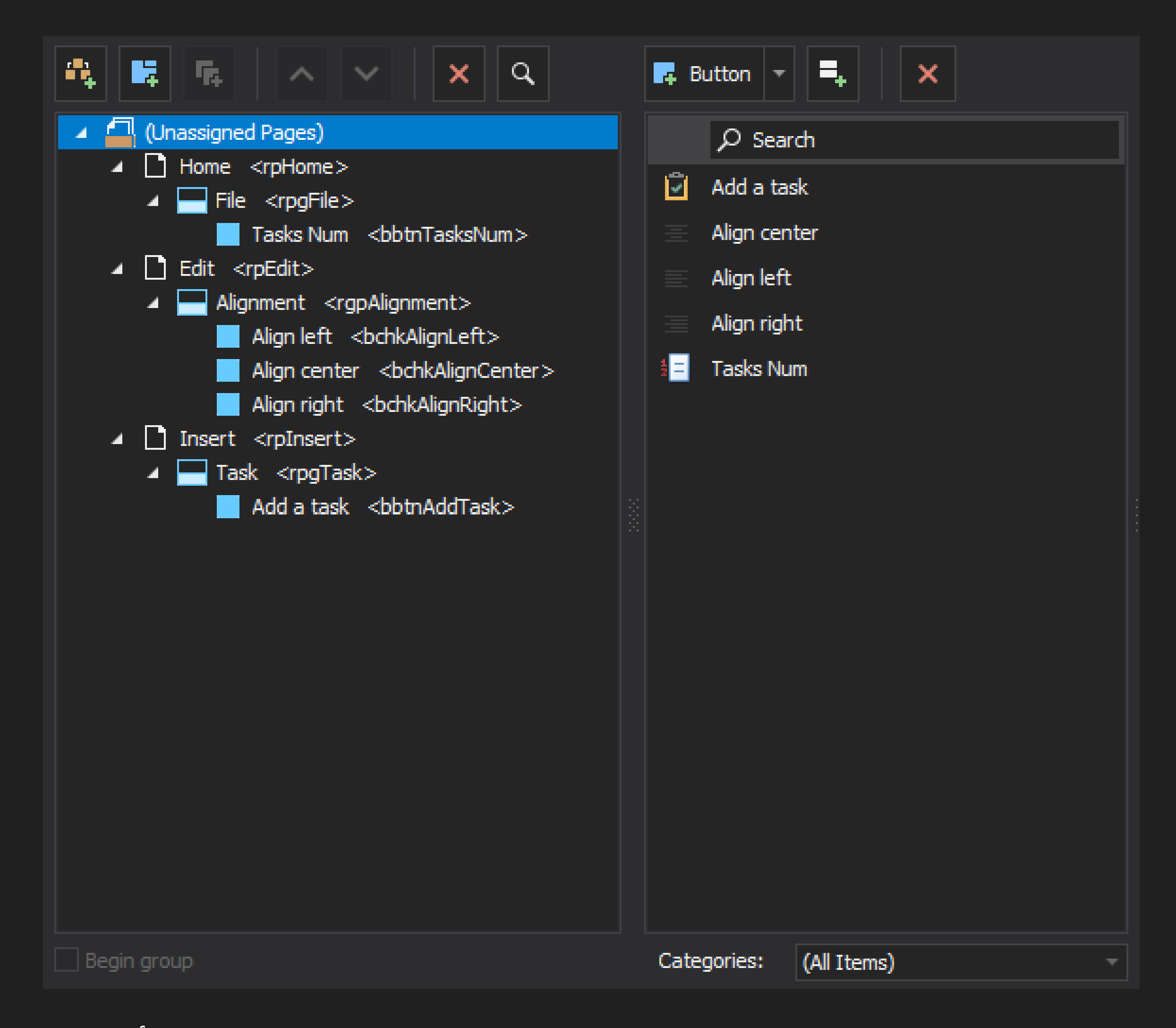
في النموذج frmTodo:



اضبط الخاصية MergeOrder للمجموعة rpgTasks على 2، ولذلك اضبط الخاصية ذاتها للمجموعات الموجودة ضمن الصفحة Edit على 0 و1 و3، حتى تحتل المجموعة rpgTasks الترتيب الثالث.

كما قم بإضافة أداة ImageCollection وسمها icDoneOrNot، ضع فيها صورتين تدل الأولى على أن المهمة قد تم إنجازها والثانية على أن المهمة لم يتم إنجازها.

في النموذج frmProject:

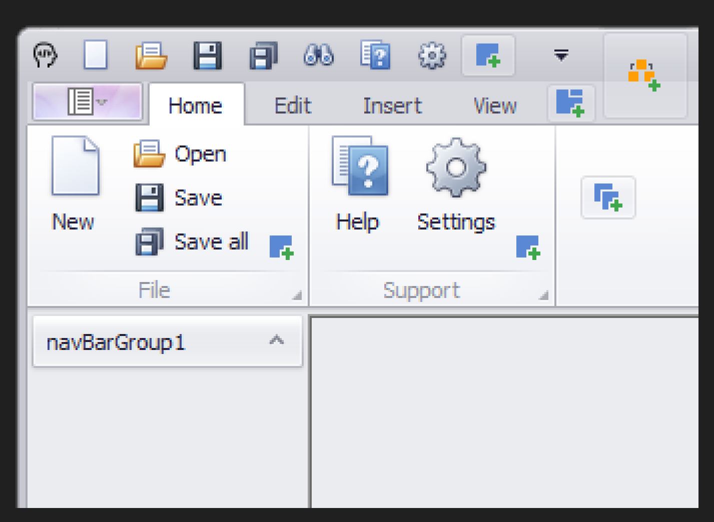


اضبط الخاصية MergeOrder للمجموعة rpgAlignment على 2، بنفس مبدأ المجموعة rpgTasks.

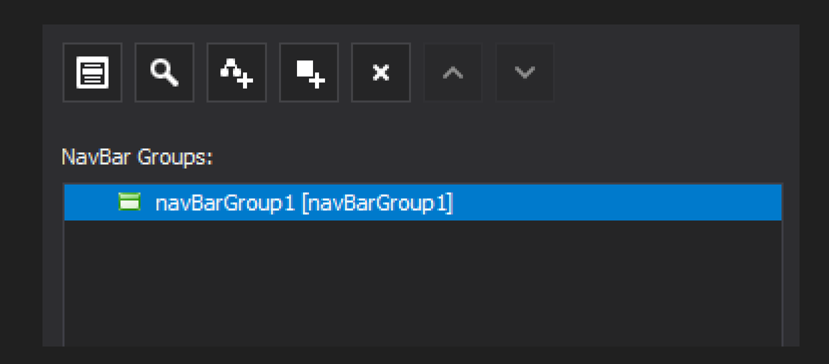
استنسخ كائنًا من أحد النموذجين الأبناء، وذلك عند حدث تحميل النافذة FormLoad على سبيل المثال، لاحظ كيف ستبدو صفحات الشريط بعد تشغيل المشروع:

الأداة NavBarControl

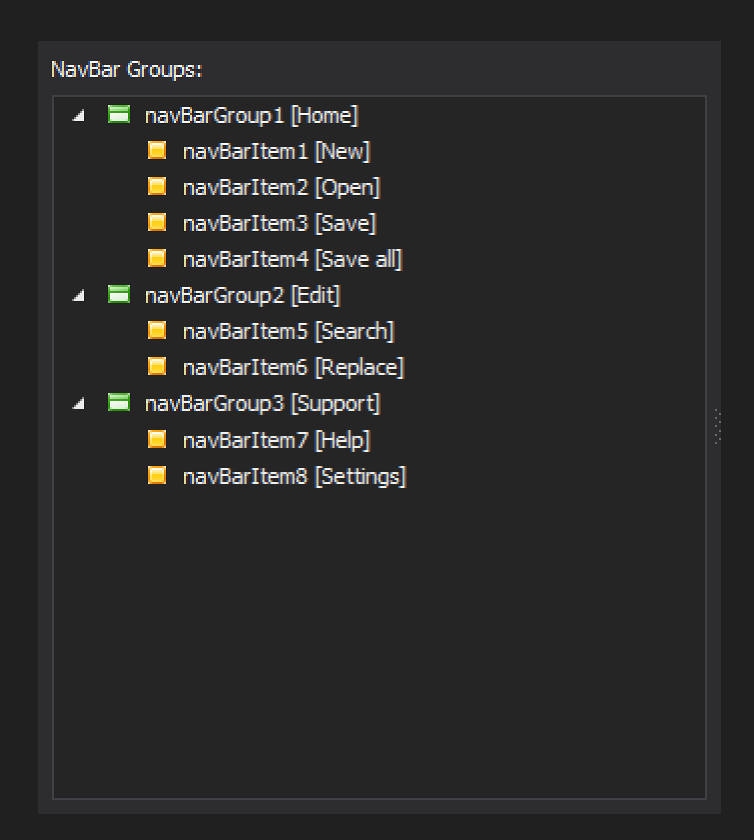
من صندوق الأدوات قم بإضافة الأداة NavBarControl واضبط الخاصية Dock لها على اليسار Left.



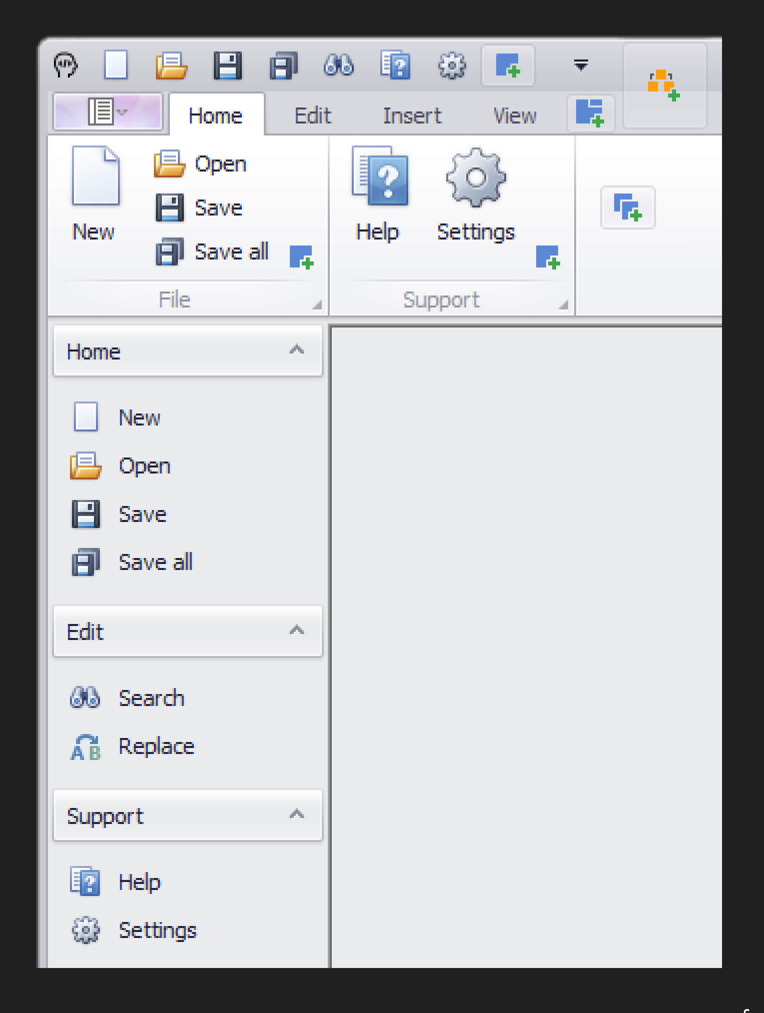
افتح المصمم الخاص بالأداة:



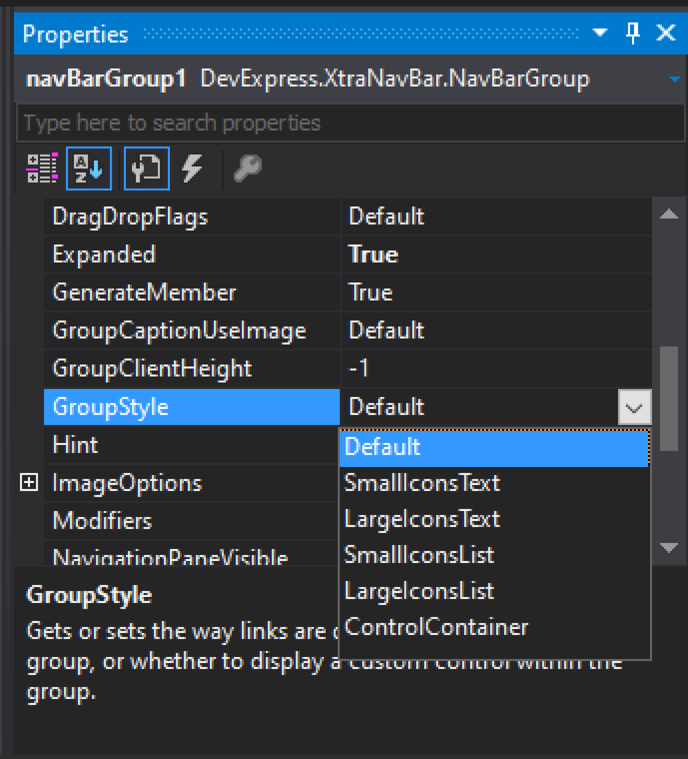
أضف المجموعات والأدوات التالية على سبيل المثال:



يمكنك تغيير مظهر أداة التنقل NavBarControl من خلال الخاصية PaintStyle. كما يمكنك تخصيص صورة لكل عنصر كما سبق.



كما يمكنك إضافة أدوات ليست من النوع NavBarItem وذلك بتغيير الخاصية GroupStyle لـ ControlContainer (أي لإضافة أدوات عادية من صندوق الأدوات ToolBox)، كما يمكنك ضبط الخاصية على قيم مختلفة لتغيير طريقة عرض العناصر:



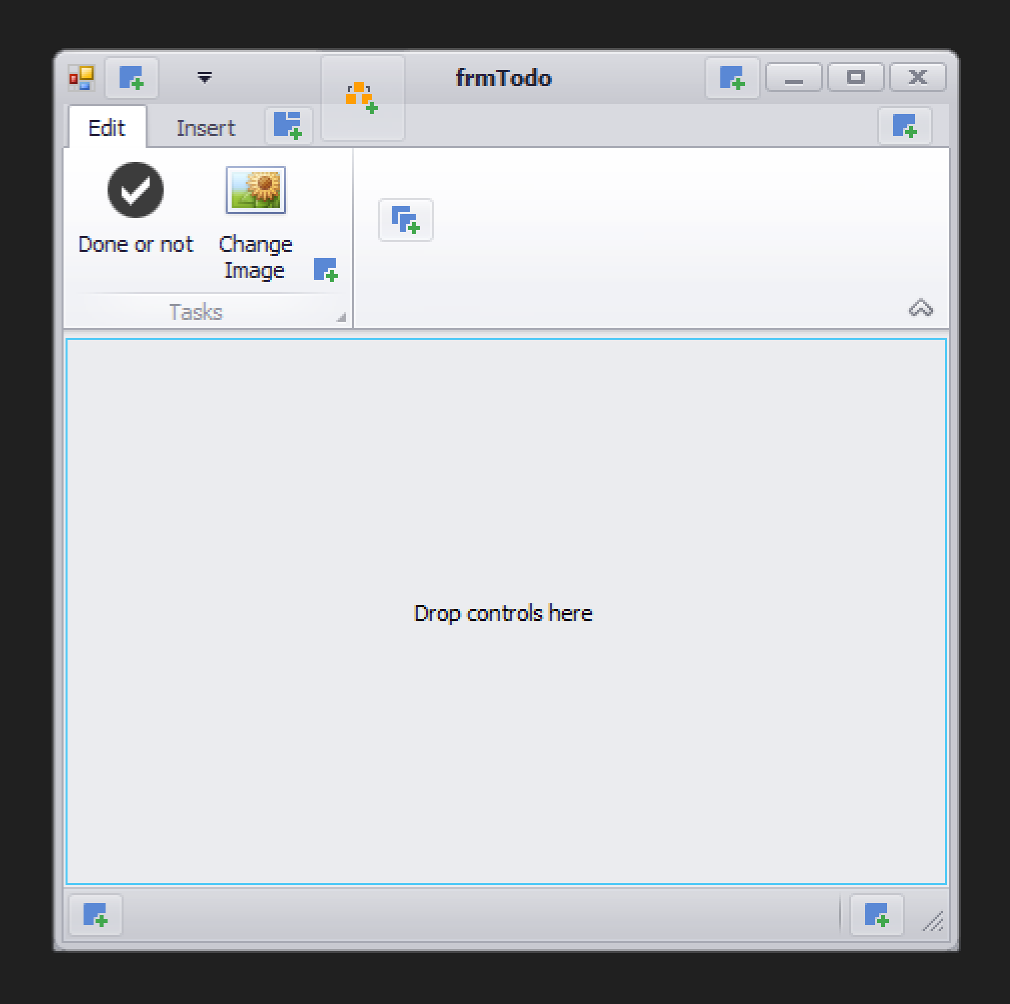
الأداة LayoutControl

تمنحك الأداة LayoutControl إمكانية ترتيب أدوات نوافذك ومحاذاتها بالنسبة لبعضها البعض وبالنسبة للأداة التي تحويها، بحيث تحافظ هذه الأدوات على ترتيبها وهوامشها حتى عند إعادة تحجيم الأداة الحاوية (أو النموذج).

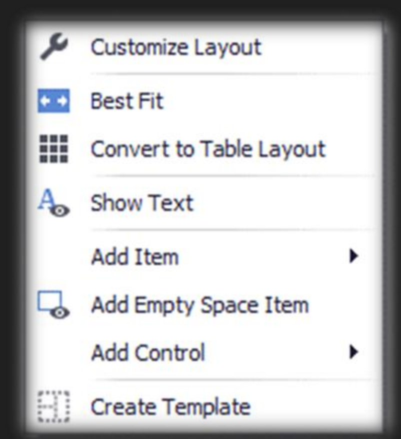
باستخدامك لهذه الأداة فإنك ستحظى بـ:

- الضبط التلقائي لنسق الأدوات.
- عدم الحاجة لضبط الأدوات وإعادة ترتيبها عند إضافة الأدوات الجديدة.
- الأدوات المدمجة لا تتداخل مع بعضها عند تغيير حجم الخط أو حجم النموذج.
- عدم انهيار النسق (ترتيب الأدوات بشكل معين) عند إضافة أداة معينة أو حذف أداة موجودة أصلًا أو تغيير ترتيبها.
- التخصيص السهل خلال وقت التصميم Design-Time.
- عرض عناوين للأدوات المدمجة.
- المحاذاة التلقائية للأدوات.
- الدعم الضمني Built-In Support للمجموعات والمجموعات المبوّبة Tabbed groups والفواصل Splitters وغيرها.
- خيارات عديدة لتخصيص الحجوم، ومحاذاة الأدوات، وظهور العناوين من عدمها، ومظهر الأدوات وغيرها.
- وضع النسق التدفقي Flow Layout Mode.
- وضع النسق المجدوَل Table Layout Mode.
- إمكانيات التخصيص خلال وقت التشغيل Run-Time من قبل المستخدم وحفظ النسق واستعادته.

من شريط الأدوات قم بإضافة الأداة LayoutControl، واضبط لها الخاصية Dock على Fill والخاصية AllowCustomization على False [1]:



انقر بالزر الأيمن على الأداة layoutControl1 (المنطقة المعنونة بـ Drop controls here) لتحصل على القائمة التالية:

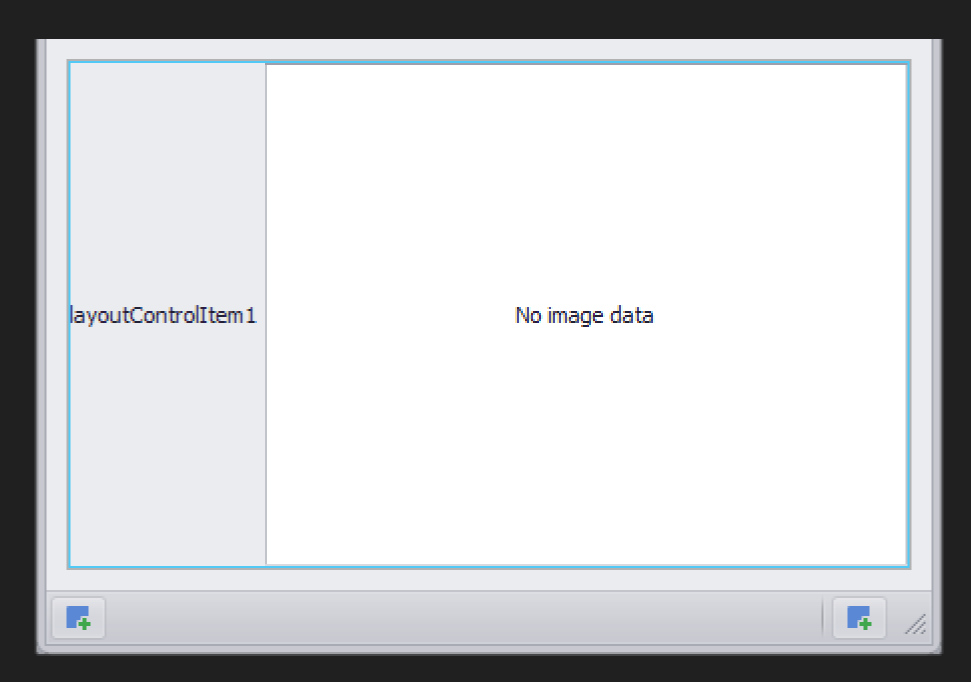


تتغير القائمة من أداة لأخرى من الأدوات الموجودة ضمن الأداة LayoutControl، ويمكنك من خلالها إضافة أدوات مختلفة من خلال القوائم Add Item وAdd Control، أو يمكنك إضافة الأدوات ببساطة من خلال سحبها من

[1] الخاصية AllowCustomization تتيح للمستخدم إعادة ترتيب أدوات LayoutControl، ولكن لتفعيلها يجب أن تضمن أن التطبيق سيقوم بتخصيص الأداة تمامًا كما اختارها المستخدم في كل مرة يعمل فيها التطبيق! غير ذلك: ضحك على اللحى.

صندوق الأدوات ToolBox، حيث تسمى عملية تصميم النافذة بتخصيص النسق على الطاير 😃 (ماعم امزح والله، تسمى هذه العملية On-the-Fly Layout Design-Time Customization).

انقر بالزر الأيمن، ثم اختر Add Control، ثم PictureEdit، لتحصل على أداة اسمها layoutControlItem1 فيها قسمان، الأداة التي اخترتها (الصورة) وعنوان لها:



انقر بالزر الأيمن على العنوان، أو بالزر الأيسر على أداة محرر الصورة ثم على السهم الذي سيظهر أعلى يسارها، لتظهر قائمة مغايرة للقائمة السابقة، اختر منها الأمر Hide Text.

افتح القائمة المنبثة مجددا – عن طريق السهم أعلى يسار الأداة – واختر الأمر إضافة عنصر منطقة خالية Add Empty Space Item:

أضف منطقة خالية أخرى، واسحبها لتصبح على يمين أداة محرر الصورة:



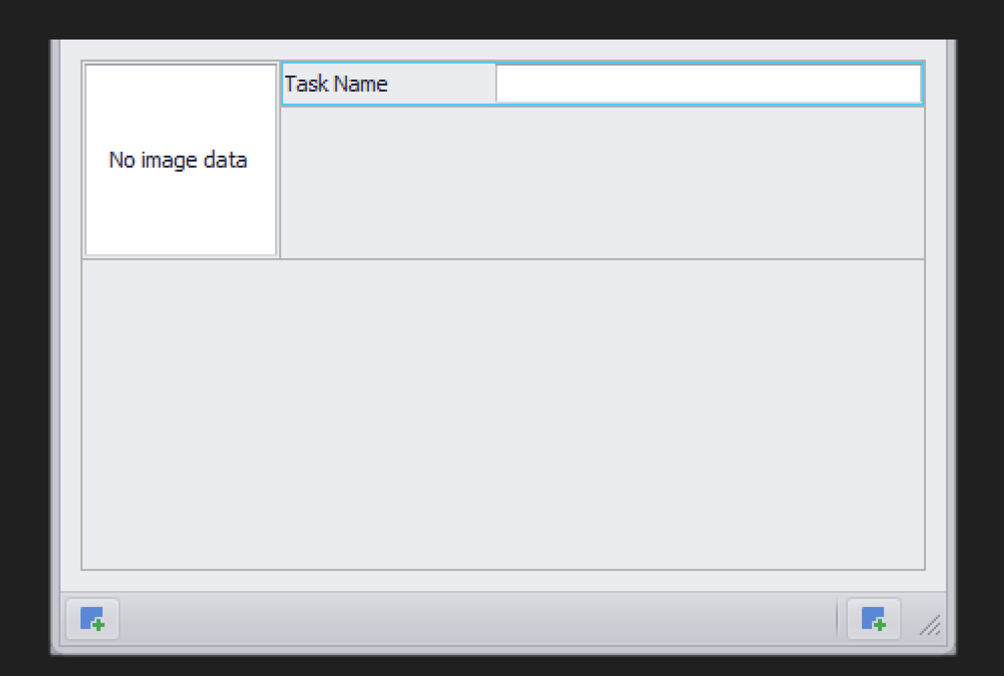
غير حجم أداة محرر الصورة، وذلك بتغيير العنصر الحاوي على الأداة، فكل عنصر يوجد عنصر يحويه (هل تذكر layoutControlItem1؟)، غير اسم هذا العنصر لـ lcPic، ثم غير حجمه لـ 100, 100 على سبيل المثال، ولا حاجة لتغيير اسم محرر الصورة pictureEdit1 على اعتبار عدم وجود مماثل له، لكن يجب ضبط خاصية SizeMode على Zoom لضمان عرض الصورة بالكامل ضمن الأداة. انقر على السهم الموجود ضمن أعلى يسار أداة pictureEdit1 واختر Size Constraints ثم Lock Size، حتى تبقى الأداة بهذا الحجم مع تغير حجم النافذة.



الآن لديك ثلاث عناصر: lcPic (والتي تحوي pictureEdit1)، وemptySpaceItem1 (الموجودة بالأسفل) وemptySpaceItem2 (الموجودة على يمين lcPic).

انقر باليمين على المنطقة الخالية الثانية ثم اختر TextEdit، اسحبها لأعلى المنطقة الخالية، ثم غير اسمها البرمجي لـ lcTaskName، وغير الاسم البرمجي للأداة TextEdit المضافة لـ txtTaskName، ثم اضبط بعض الخصائص كما يلي:

الأداة	الخاصية	القيمة
lcTaskName	Text	Task Name
	TextAlignMode	CustomSize
	TextSize	100, 20
txtTaskName	HAlignment [1]	Center



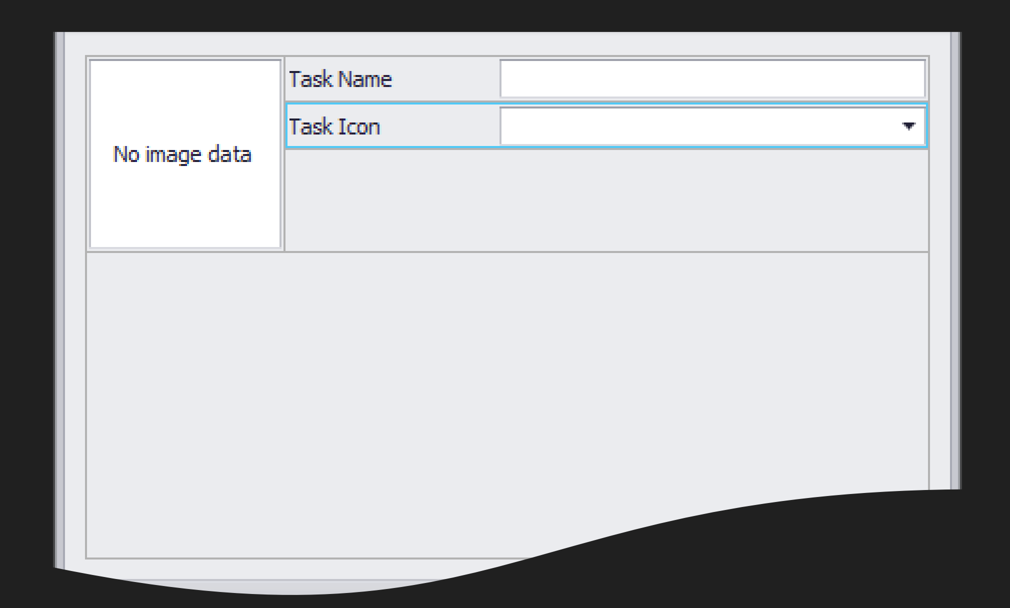
[1] هذه الخاصية هي خاصية فرعية، يمكنك الوصول إليها من الخاصية Properties ثم Appearance ثم TextOptions.

الأداة PopupGalleryEdit

تشبه هذه الأداة أداة ComboBox وأداة ListView معًا، فهي تعرض لك قائمة منسدلة (مثل أداة ComboBox) وتعطيك إمكانية تخصيص صور للعناصر وترتيب العناصر ضمن مجموعات (مثل أداة ListView).

من صندوق الأدوات ToolBox ابحث عن الأداة وأضفها للمنطقة الخالية الثانية – الموجودة بجانب أداة محرر الصورة، واضبط خصائصها كما يلي:

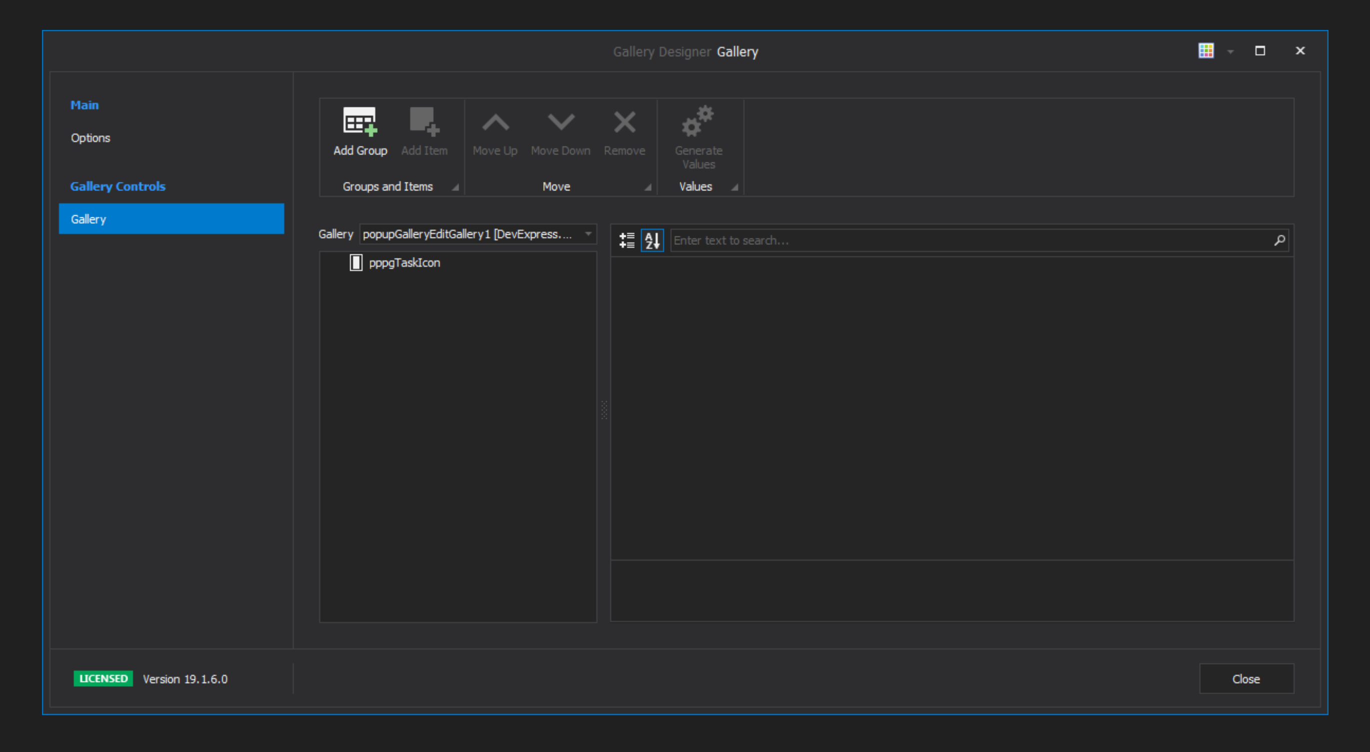
الأداة	الخاصية	القيمة
lcTaskType	Text	Task Type
	TextAlignMode	CustomSize
	TextSize	100, 20
pppgTaskType	HAlignment	Center
	ImageSize [1]	32, 32
	ShowItemText [2]	True



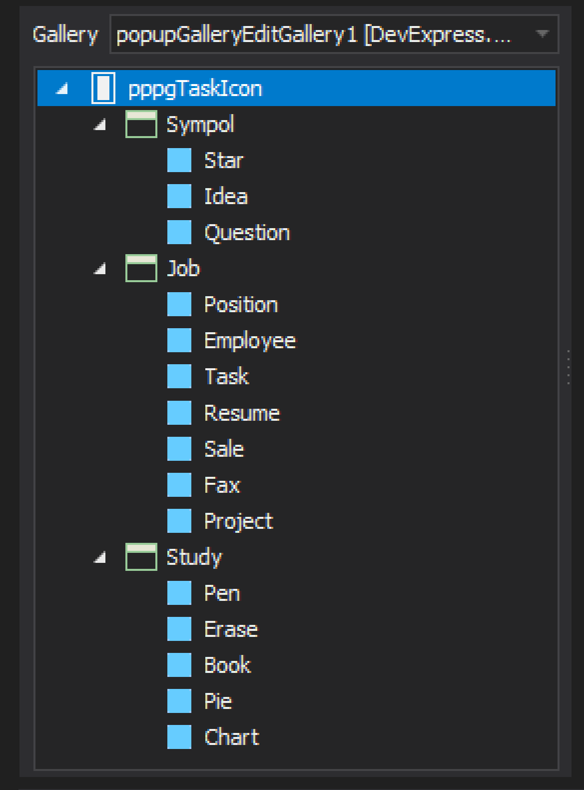
[1] هذه الخاصية هي خاصية فرعية ضمن الخاصية Gallery والتي يمكن الوصول لها من الخاصية Properties.
[2] نفس الخاصية السابقة.

افتراضيًّا يجب أن تكون الخاصية ItemCheckMode لها القيمة SingleCheck، لكن قد تجد في نسخة أخرى قيمة افتراضية مغايرة، وعليه تحقق من أن قيمة هذه الخاصية هي SingleCheck.

انقر على اختصار الأوامر المميزة Smart Tag الخاص بأداة PopupGallery، ثم تعديل Edit Gallery ثم انتقل لـ Gallery:

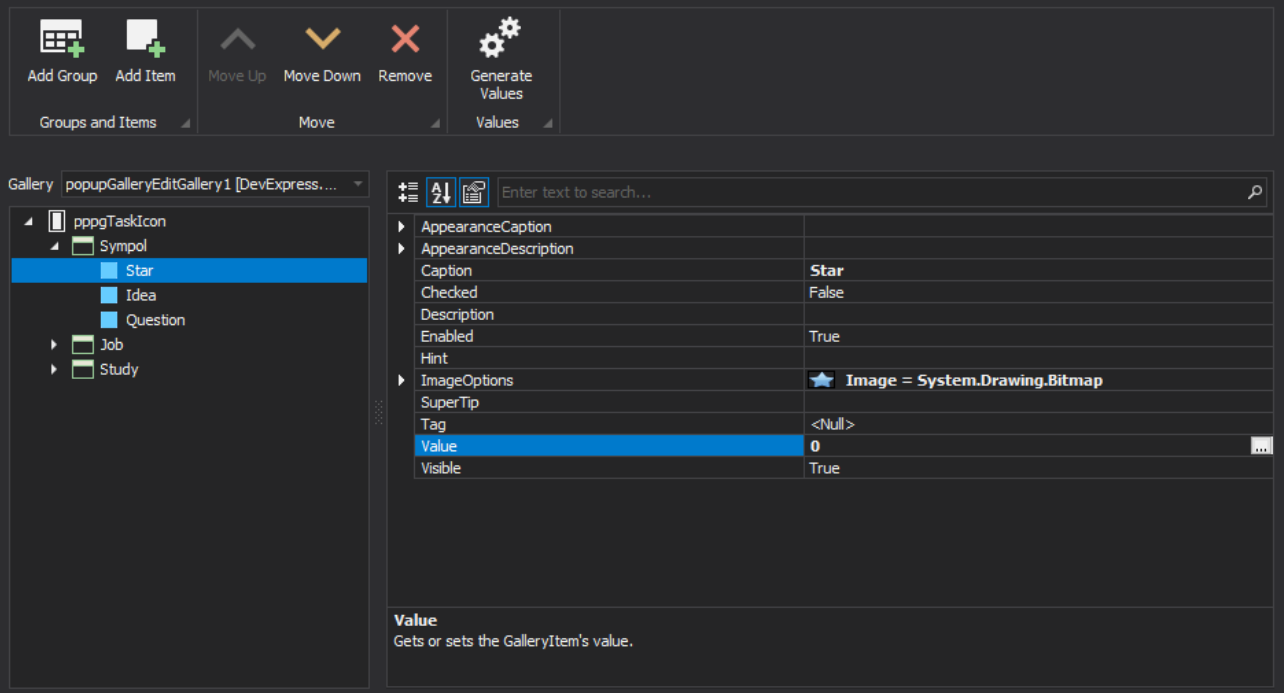


لاحظ أن المعرض خال لا يحوي أي مجموعة ولا عناصر، وما سنقوم به في هذه الفقرة هو إضافة المجموعات والعناصر التي أضفناها في الصفحة rgEdit ضمن الأداة rgbIcon من النموذج الأم.



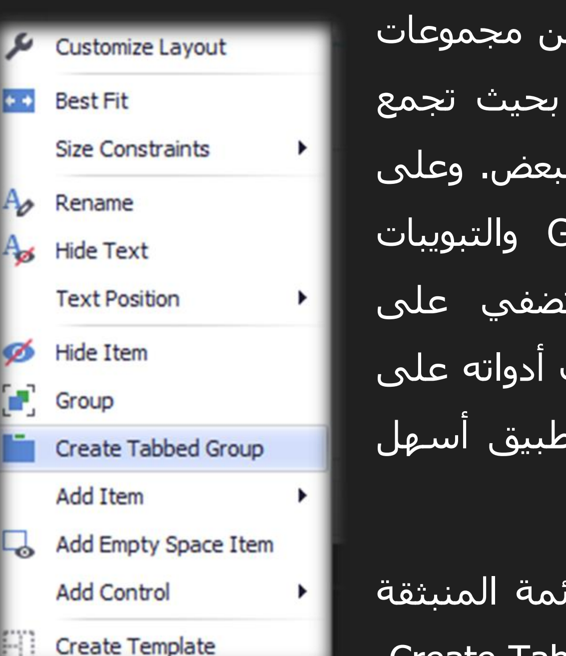
قم بإضافة نفس العناصر والمجموعات التي أضفناها ضمن الفقرة **تهيئة النموذج الأم في المشروع، بعمق**، وهي موضحة بالصورة المجاورة.

انقر على العنصر الأول لعرض خصائصه، عدل الخاصية Value واجعلها 0، ثم انقر على Generate Values:



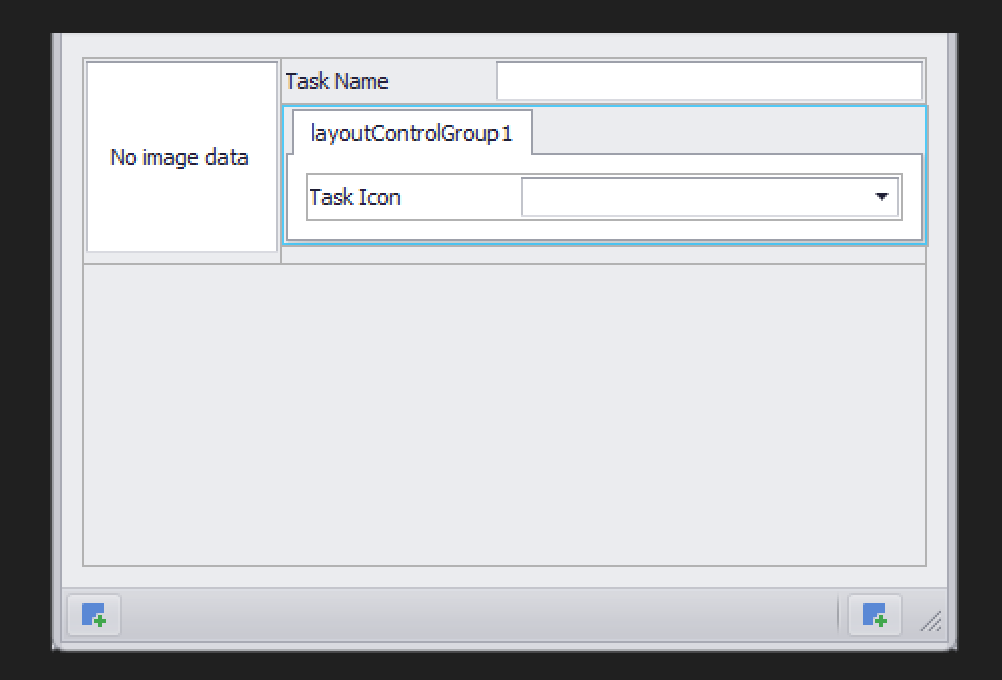
(لا تنس النقر على Generate Values في الأعلى)

الأداة LayoutControl مجددًا، المجموعات والتبويبات



يمكنك تنظيم محتوى أداة LayoutControl ضمن مجموعات Groups ومجموعات مبوبة Tabbed groups، بحيث تجمع الأدوات التي لها وظائف متشابهة مع بعضها البعض. وعلى اعتبار أن أدوات المجموعات GroupControls والتبويبات TabbedControls واللوائح PanelControls تضفي على برامجك ميزة تنظيم محتواه وترتيبه وفق وظائف أدواته على شكل فقرات، فإن هذا سيجعل استخدام التطبيق أسهل للمستخدم.

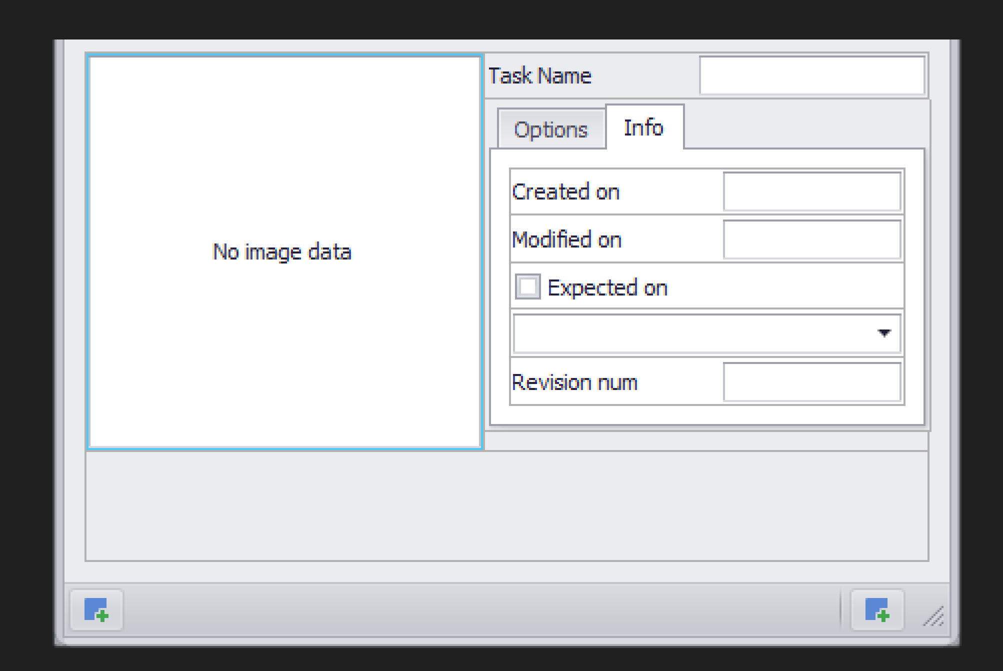
انقر على lcTaskIcon بالزر الأيمن لتظهر القائمة المنبثقة الموضحة بالشكل المجاور ثم اختر Create Tabbed Group، لتحصل على الأداة tabbedControlGroup1 وبداخلها الأداة layoutControlGroup1:



غيّر الاسم البرمجي للمجموعة لـ lcgOptions، وخاصية Text لها للقيمة "Options"، ثم انقر بالزر الأيمن على tabbedControlGroup1 واختر الأمر Add Tab، ثم غير اسمها البرمجي لـ lcgInfo وخاصية Text لها للقيمة "Info". ثم أضف للمجموعة ثلاثة أدوات

TextEdit واضبط خصائصها وخصائص حاوياتها كما سبق، ثم أضف أداة CheckEdit وأداة DateEdit وفق الجدول التالي:

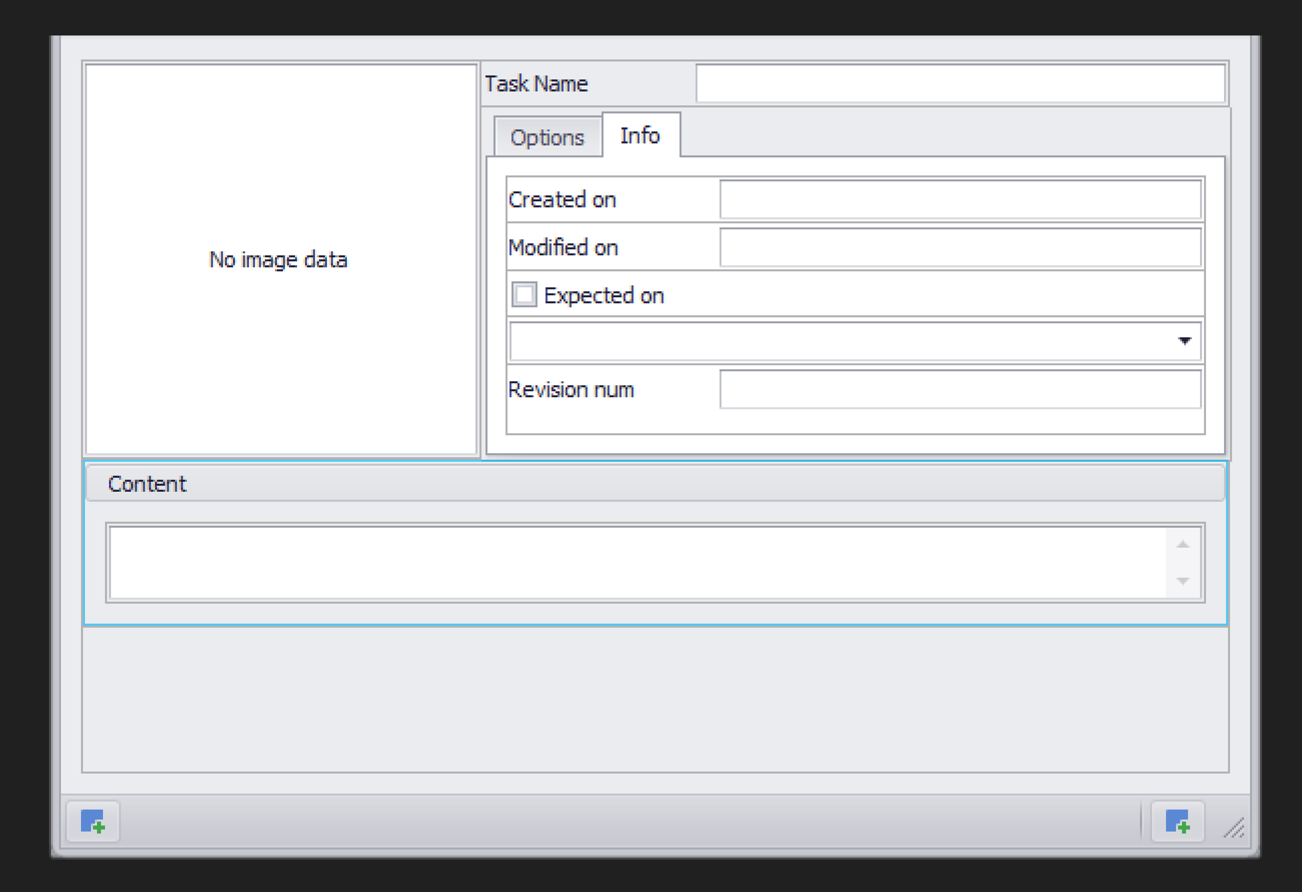
<table>
<tr><th>الأداة</th><th>الخاصية</th><th>القيمة</th></tr>
<tr><td>chkExpectedOn</td><td>Text</td><td>Expected on</td></tr>
<tr><td rowspan="3">dteExpectedOn</td><td>Enabled</td><td>False</td></tr>
<tr><td>HAlignment</td><td>Center</td></tr>
<tr><td colspan="2">انقر على lcExpectedOn (الأداة الحاوية لهذه الأداة) ثم اختر Hide Text</td></tr>
<tr><td rowspan="2">lcPic</td><td>Size</td><td>200, 200</td></tr>
<tr><td colspan="2">قبل تغيير الحجم انقر على السهم أعلى يسار الصورة pictureEdit1 ثم من Size Constraints اختر استعادة الافتراضيات، لتتمكن من تغيير الحجم
ثم أعد إقفال الحجم مجددًا</td></tr>
</table>



احذف المنطقة الخالية الثانية emptySpaceItem2 (الجزء الصغير أسفل المجموعة المبوبة tabbedControlGroup1) لأننا لن نحتاجها بعد الآن، وانقر على المنطقة الخالية الأولى بالزر الأيمن، ثم أضف الأداة MemoEdit وغير اسم حاويتها البرمجي لـ

lcTaskContent وحاذي محتواها النصي نحو المنتصف، عدّل حجم النافذة frmTodo لتصبح منطقة التصميم أوضح:

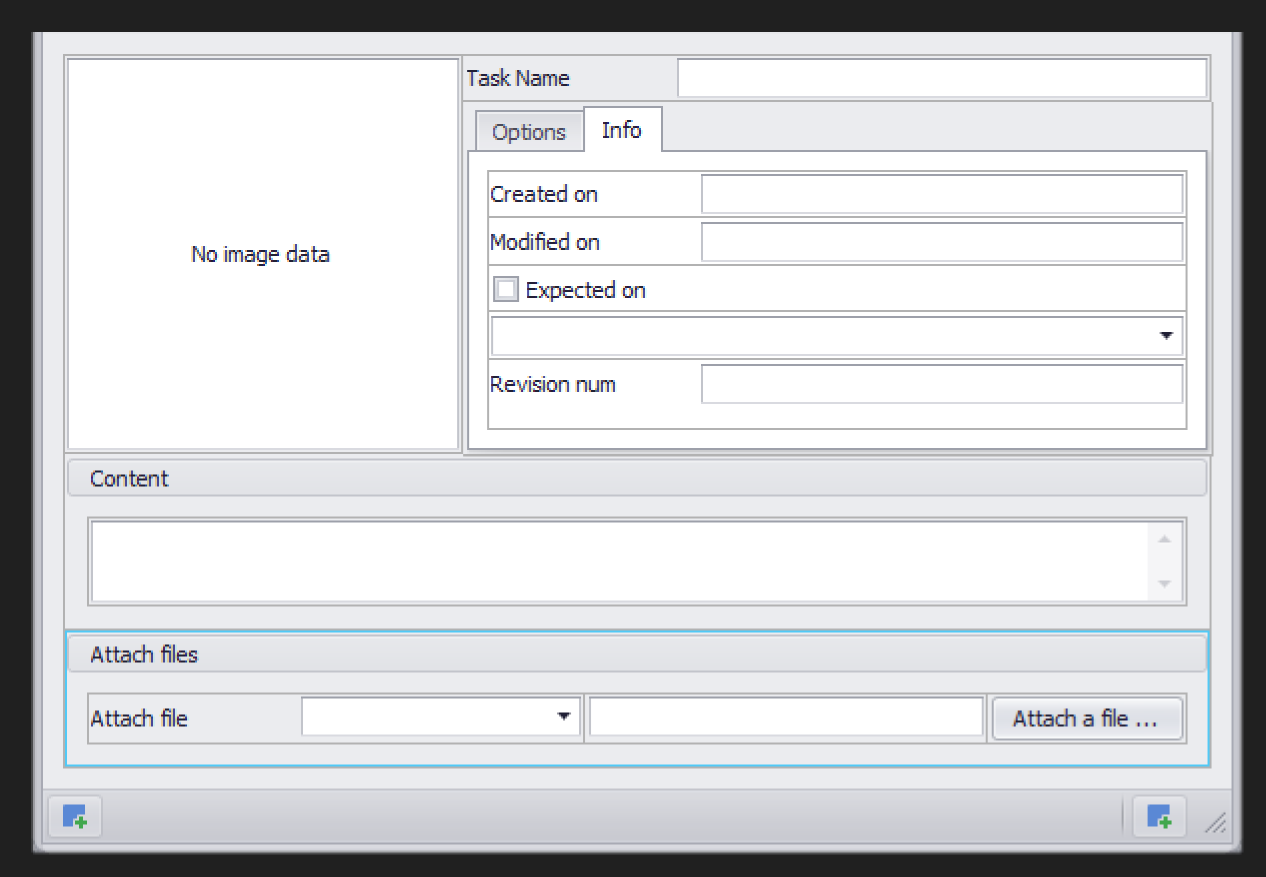
انقر بالزر الأيمن على lcTaskContent (في الصورة السابقة layoutControlItem1، فقد تم أخذها قبل تغيير اسمها) ثم انقر على Group لإضافة الأداة لمجموعة (غيّر الاسم البرمجي لها لـ lcgTaskContent والخاصية Text لـ "Content")، ثم انقر مجددًا على lcTaskContent واختر Hide Text. وغير الخاصية GroupStyle الخاصة بالمجموعة lcgTaskContent للقيمة Title:



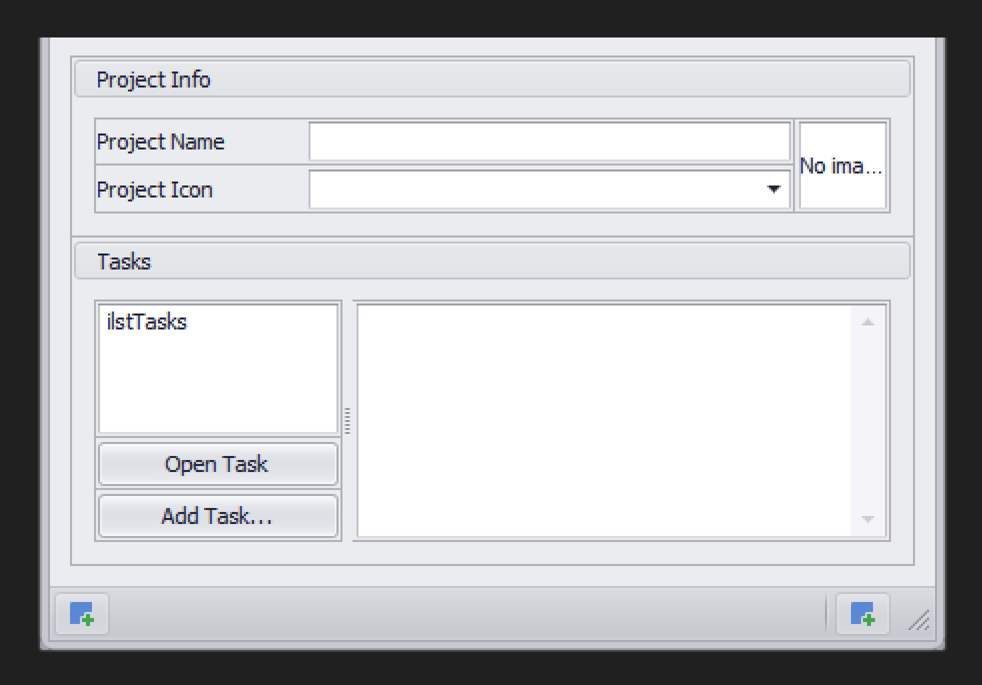
وأخيرًا، سنقوم بإضافة فقرة [1] للبرنامج لربط ملفات البرنامج مع روابط مختلفة، ولكن هذه المرة سنضيف المجموعة أولًا ثم سنضيف لها أدوات ربط ملفات البرنامج مع غيره. لذلك فانقر بالزر الأيمن على المنطقة الخالية الأولى ثم اختر Group، لتصبح هذه المنطقة الخالية بداخل المجموعة.

غيّر الاسم البرمجي للمجموعة لـ lcgAttachFiles والخاصية Text لها لـ "Attach files"، والخاصية GroupStyle لـ Title. قم بإضافة أداة ComboBoxEdit وأداة TextEdit وأداة SimpleButton، وقم بإخفاء نصوص حاويات صندوق النص والزر. غير الأسماء البرمجية لهذه الأدوات لـ cmbAttached وtxtAttached وbtnAttachFile، ثم اقفل الأداتين الأولى والأخيرة، وذلك كما يلي:

[1] تدعى المجموعات Groups في البرامج أحيانًا بالفقرات.



انتقل الآن للنافذة frmProject، صممها بالشكل التالي كما سبق:



استعن بالجدول التالي لضبط خصائص النافذة:

الأداة	الخاصية	القيمة
layoutControl1	AllowCustozimation	False
	Dock	Fill
lcgProjectInfo	GroupStyle	Title
	Text	Project Info
lcProjectName	Text	Project Name
	TextAlignmentMode	CustomSize
	TextSize	100, 20
lcProjectIcon	Text	Project Icon
	TextAlignmentMode	CustomSize
	TextSize	100, 20
lcPic	Size	48, 48
	اضبط Size Constraints على وضع إقفال الأداة	
pppgProjectIcon	اضبطها تمامًا كما في النافذة frmTodo	
lcgTasks	GroupStyle	Title
	Text	Tasks
ilstTasks	ImageList	icDoneOrNot
sbtnOpenTask	Text	Open Task
sbtnAddTask	Text	Add Task...
memoEdit1	HAlignment	Center
splitterItem1	افتراضيًّا، الأداة ستُضاف بشكل أفقي، اسحبها لتصبح بشكل شاقولي بين أداة ilstTasks وmemoEdit1	

التحكم بأدوات المشروع من خلال الكود

يمكنك تخصيص الأدوات التي ستظهر في شريط الوصول السريع QAT، كما يمكنك عرض وإخفاء الأدوات على أساس الملفات المفتوحة. الكود التالي يلغي إظهار بعض الأزرار ويختار عنصرًا من الأداة rgbIcon: [1]



```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Windows.Forms;
using DevExpress.XtraBars;
using DevExpress.XtraEditors;
using DevExpress.XtraBars.Ribbon;
using System.IO;

namespace DevExpressTest
{
    public partial class Form1 : RibbonForm
    {
        public Form1()
        {
            InitializeComponent();
        }

        frmTodo ft;
        frmProject fp;

        private void Form1_Load(object sender, EventArgs e)
        {
            // بنفس طريقة الأكواد التالية، حاول أتمتة العملية بحيث تُظهر الأدوات التي اختارها المستخدم من الإعدادات
            ribbonControl1.Toolbar.ItemLinks[3].Visible = false;
            ribbonControl1.Toolbar.ItemLinks[4].Visible = false;
            ribbonControl1.Toolbar.ItemLinks[6].Visible = false;

            rgbIcon.Gallery.Groups[0].Items[3].Checked = true;
        }
    }
}
```

المراجع المستخدمة بالكود السابق ستحتاجها بالأمثلة اللاحقة، والكائنين ft و fp كذلك.

لاحظ التطبيق عند تشغيله:



[1] يمكنك – بالاعتماد على فهرس index الأدوات – السماح للمستخدم باختيار العناصر التي سيتم تنفيذ عمليات بدء تشغيل التطبيق عليها، وذلك بإضافة هذه الإمكانية ضمن قسم الإعدادات في البرنامج.



يمكن للمستخدم تغيير العناصر المعروضة ضمن شريط الوصول السريع، لكنه سيفقد تغييراته عند إيقاف التطبيق. ولذلك إن أردت إعطاء المستخدم إمكانية تخصيص هذا الشريط – وغيره – يمكنك حفظ فهرس الأدوات التي يرغب المستخدم بعرضها، وإخفاء كل ما سواها عند تحميل التطبيق.
هذا وبعض أدوات ديف إكسبريس تتمتع بإمكانية تخصيص الأدوات، بحيث يمكن للمستخدم تخصيص عناصر أي أداة أثناء طور التشغيل Run-Time.

وإذا أردت تخصيص العناصر والأدوات بناءً على نوع النافذة الابن الفعالة (أو على وجود نوافذ أو لا)، طور الكود التالي:

```
private void Form1_MdiChildActivate(object sender, EventArgs e)
{
    Form f = ActiveMdiChild;
    if (f is frmTodo)
        //الأكواد التي ترغب بتنفيذها عندما تكون النافذة الابن الفعالة هي نافذة المهام
        //...
    else if (f is frmProject)
        //الأكواد التي ترغب بتنفيذها عندما تكون النافذة الابن الفعالة هي نافذة المشاريع
        //...
    else
        //الأكواد التي ترغب بتنفيذها عندما لا تكون هناك نافذة ابن فعالة
        //...
}
```

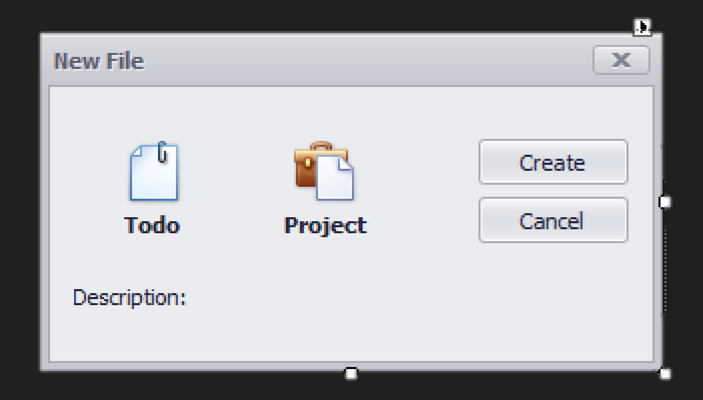
هل تذكر الواجهات Interfaces في الفصل الخامس؟ يمكنك تطوير فئات نوافذ هذا المشروع – خصوصًا إن كان هناك العشرات منها – لهذا الغرض.

من الضروري أن تضبط عناصر وأدوات التطبيق لتناسب السياق الجاري التعامل معه، فعند عدم وجود نوافذ أبناء، لا حاجة لظهور (أو تفعيل) أزرار البحث والحفظ والحذف وغيرها، وقد تكون بعض الأدوات لا حاجة لها حتى ولو كان هناك نوافذ أبناء يتم التعامل لها، وذلك حسب السياق وما يناسبه من عمليات.

الأمر جديد New

يمكن للمستخدم إنشاء ملفات المشروع (ملف المهمة Todo وملف المشروع Project) من خلال الأمر New، وينبغي على هذه النافذة أن تكون بسيطة غير معقدة.

أضف نموذجًا من النوع XtraForm وصممه بالشكل التالي:



تحوي النافذة أربعة أزرار SimpleButton ولائحة LabelControl، وذلك كما هو موضح بالجدول التالي:

الأداة	الخاصية	القيمة
frmNew	FormBorderStyle	FixedSingle
	MaximizeBox	False
	MinimizeBox	False
	ShowIcon	False
	ShowInTaskbar	False
	Size	311, 169
	StartPosition	CenterParent
	Text	New File
lblDesc	Location	12, 98
	Text	Description:
SbtnNewTodo	Font.Bold	True

<table>
<tr><td></td><td>Image [1]</td><td rowspan="8"></td></tr>
<tr><td>TopCenter</td><td>ImageToTextAlignment [2]</td></tr>
<tr><td>MiddleCenter</td><td>Location [3]</td></tr>
<tr><td>12, 12</td><td>Location</td></tr>
<tr><td>Light</td><td>PaintStyle</td></tr>
<tr><td>80, 80</td><td>Size</td></tr>
<tr><td>Todo</td><td>Text</td></tr>
<tr><td colspan="2">بالمثل الأداة sbtnNewProject لكن موقعها 98, 12</td></tr>
<tr><td>OK</td><td>DialogResult</td><td rowspan="4">sbtnCreate</td></tr>
<tr><td>214, 26</td><td>Location</td></tr>
<tr><td>Create</td><td>Text</td></tr>
<tr><td colspan="2">بالمثل الأداة sbtnCancel لكن موقعها 214, 55 عدا الخاصية DialogResult اتركها None</td></tr>
</table>

استخدم هذا الكود:



```
using System;
using System.Windows.Forms;
using DevExpress.XtraEditors;

namespace DevExpressTest
{
    public partial class frmNew : XtraForm
    {

        public frmNew()
        {
            InitializeComponent();
            lblDesc.Text = "Description: Create a task file.";
        }

        public enum FileToCreate
        {
            TodoFile, ProjectFile
        }
```

[1] تجدها ضمن الخاصية ImageOptions.
[2] نفس الخاصية السابقة.
[3] نفس الخاصية السابقة.

```
        public FileToCreate ftc = FileToCreate.TodoFile;

        private void sbtnNewTodo_Click(object sender, EventArgs e)
        {
            lblDesc.Text = "Description: Create a task file.";
            ftc = FileToCreate.TodoFile;
        }

        private void sbtnNewProject_Click(object sender, EventArgs e)
        {
            lblDesc.Text = "Description: Create a project file.";
            ftc = FileToCreate.ProjectFile;
        }

        private void sbtnNewTodo_DoubleClick(object sender, EventArgs e)
        {
            lblDesc.Text = "Description: Create a task file.";
            ftc = FileToCreate.TodoFile;
            this.DialogResult = System.Windows.Forms.DialogResult.OK;
            Close();
        }

        private void sbtnNewProject_DoubleClick(object sender, EventArgs e)
        {
            lblDesc.Text = "Description: Create a project file.";
            ftc = FileToCreate.ProjectFile;
            this.DialogResult = System.Windows.Forms.DialogResult.OK;
            Close();
        }

        private void sbtnCreate_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void sbtnCancel_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

لعلّك لاحظت – لو أنك قرأت محتويات جدول الخصائص – أننا غيّرنا قيمة الخاصية DialogResult للزر sbtnCreate وجعلناها OK، وعليه فإن نتيجة النافذة ستصبح OK. كما أني أظن أنه خطر ببالك الآن أننا سنفتح النافذة كمربع حوار – عن طريق الطريقة ShowDialog – والتحقق من أن نتيجة النافذة هي OK، عندها يتم إنشاء ملف جديد. لكن ماذا عن نوع الملفات؟ علامَ سيعتمد البرنامج ليعرف نوع الملف المطلوب إنشاؤه؟ لعلّك – أيضًا – لاحظت وجود المعدّد FileToCreate، واسمه يخبرك بوظيفته، هو معدد يأخذ قيمتين: ملف Todo أو ملف Project!

استخدم الكود التالي في النافذة الرئيسية من التطبيق Form1 لإنشاء ملفات جديدة:



```
private void bbtnNew_ItemClick(object sender,
DevExpress.XtraBars.ItemClickEventArgs e)
{
    frmNew fn = new frmNew();
    if (fn.ShowDialog() == System.Windows.Forms.DialogResult.OK)
        if (fn.ftc == frmNew.FileToCreate.TodoFile)
        {
            ft = new frmTodo();
            ft.MdiParent = this;
            ft.Text = "New todo file";
            ft.Show();
        }
        else
        {
            fp = new frmProject();
            fp.MdiParent = this;
            fp.Text = "New project file";
            fp.Show();
        }
}
```

لاحظ الكائن fn المستنسخ عن النافذة frmNew، والذي اعتمدنا عليه للوصول لمتغيرات المعدّد FileToCreate والذي جعلنا إمكانية الوصول إليه عامة public، واعتمادًا على هذا المتغير يمكن للبرنامج معرفة ما يرغب المستخدم بإنشاءه!

أما إذا أردت أسلوبًا محترمًا أكثر لنقل البيانات بين النوافذ، فاعتمد على الفئات الوسيطة، للمزيد راجع الملحق د.

الأمر حفظ Save

عند النقر على أزرار الحفظ Save يتم التحقق أولًا من وجود نافذة ابن MdiChild فعّالة، ثم يتم حفظ الملف الفعّال – المختار – وفق خوارزمية تناسب نوعه، والكود المستخدم لذلك هو من الشكل:



```
private void bbtnSave_ItemClick(object sender, ItemClickEventArgs e)
{
    Form f = ActiveMdiChild;
    if (f != null)
        if (f is frmTodo)
            SaveTodoFile();
        else
            SaveProjectFile();
}
```

أما محتويات الطرق SaveTodoFile وSaveProjectFile فأتركها لك 🤓. ولكن أود إيضاح نقطة، وهي عند حفظ الملف لأول مرة فإن النافذة لا تتعلق بأي ملف على القرص الصلب (على عكس حالة فتح ملف محفوظ، فإن حقول وفقرات النافذة ستُملأ على أساس ملف ما موجود على القرص الصلب). لذلك فإنه ينبغي عليك تطوير كود يقوم بحفظ الملف الجاري حفظه في ملف على القرص الصلب إذا لم يكن محفوظًا مسبقًا.



```
void SaveProjectFile()
{
    //Codes
    //...
    if (fileIsNotSaved)
    {
        XtraSaveFileDialog xsave = new XtraSaveFileDialog();
        xsave.Filter = "Project files (*.prj)|*.prj";
        if (xsave.ShowDialog() == System.Windows.Forms.DialogResult.OK)
        {
            CreateTheFile();
        }
    }
    //...
    //Codes
}

void SaveTodoFile()
{
    //Codes
    //...
    if (fileIsNotSaved)
    {
        XtraSaveFileDialog xsave = new XtraSaveFileDialog();
        xsave.Filter = "Task files (*.tsk)|*.tsk";
        if (xsave.ShowDialog() == System.Windows.Forms.DialogResult.OK)
        {
            CreateTheFile();
        }
    }
    //...
    //Codes
}
```

وعلى اعتبار أن محتوى النافذة – سواءً أكانت نافذة مهام frmTodo أو نافذة مشاريع frmProject – سيحفظ بالكامل ضمن ملف واحد فإن الصورة يجب حفظها ضمن هذا الملف، ولذلك يمكنك تحويل الصورة لقيمة نصية وتخزينها ضمن الملف، والعكس عند أخذها من الملف يتم تحويلها من قيمة نصية لصورة، وذلك بالكود التالي:



```
public Image Base64ToImage(string base64String)
{
    // تحويل القيمة النصية لمصفوفة بايتات
    byte[] imageBytes = Convert.FromBase64String(base64String);
    // تحويل مصفوفة البايتات لصورة
    using (var ms = new MemoryStream(imageBytes, 0, imageBytes.Length))
    {
        Image image = Image.FromStream(ms, true);
        return image;
    }
}

public string ImageToBase64(Image image, ImageFormat format)
{
    using (MemoryStream ms = new MemoryStream())
    {
        // تحويل الصورة لمصفوفة بايات
        image.Save(ms, format);
        byte[] imageBytes = ms.ToArray();

        // تحويل مصفوفة البايتات لقيمة نصية
        string base64String = Convert.ToBase64String(imageBytes);
        return base64String;
    }
}
```

أو يمكنك التعامل معها كمصفوفة بايتات، وهو أفضل أسلوب للتعامل مع أي نوع من الملفات، كما يمكنك الاعتماد على الأداة FileModel من الفصل السابع للتعامل مع الملفات لقراءتها وكتابتها.

الأمر حفظ الكل SaveAll

عند النقر على حفظ الكل SaveAll يتم حفظ جميع النوافذ الأبناء MdiChildren، وذلك باستدعاء الطريقة الخاصة بنوع كل ملف من الملفات المفتوحة. تأمل الكود:



```
private void bbtnSaveAll_ItemClick(object sender, ItemClickEventArgs e)
{
    foreach (Form f in MdiChildren)
        if (f is frmTodo)
            SaveTodoFile();
        else
            SaveProjectFile();
}
```

الأمر فتح Open

عند فتح الملفات تُفتح النافذة المناسبة للملف المفتوح، وذلك حسب نوع الملف الذي تم فتحه من الأداة XtraOpenFileDialog (أو أداة فتح الملفات القياسية). وكما ذكرنا سابقًا فإن الملفات ستُحفظ بصيغتين، وعلى أساس صيغة الملف المفتوح سيتم فتح النافذة المناسبة لهذا الملف (frmTodo أو frmProject) وإملاء حقولها ببيانات هذا الملف، وفق الخوارزمية التي قمتَ بتخزين البيانات على أساسها في هذا الملف.

الأمر بحث Search

الفكرة من هذا الأمر بهذا المشروع هي عند وجود ملفات كثيرة مفتوحة ويُراد البحث عن معلومة ما ضمن الملفات المفتوحة، فإنه يتم جعل النافذة التي فتحت الملف الذي تم إيجاد نتيجة البحث فيه نافذة فعالة ActiveMdiChild. والفكرة منها باختصار تتمثل بالكود التالي:

```
private void bbtnSearch_ItemClick(object sender, ItemClickEventArgs e)
{
    string search = "قيمة تحصل عليها من نافذة البحث";
    string controlName = "اسم الأداة التي سيتم البحث فيها";
    Form form = ...;
    foreach (Form f in MdiChildren)
        if (f is form)
            foreach (Control c in f.Controls.Find(controlName, true)[0])
                if (c.Text == search)
                {
                    //Codes
                    //...
                }
}
```

حيث ينبغي عليك أولًا تصميم نافذة حوار Dialog – مثل نافذة جديد New – وعلى أساسها يتم تحديد قيمة المتغير search، ثم يتم البحث في جميع النماذج الأبناء، ومن أجل جميع النماذج الأبناء يتم البحث ضمن الأداة الفلانية controlName – والتي تُحدَّد على أساس ما سيختاره المستخدم من نافذة الحوار frmSearch – على القيمة الفلانية search.

الخاصية Controls عبارة عن مجموعة Collection تمثل الأدوات المحضونة ضمن أعلى مستوى، أي أنها ستعيد الأدوات المحضونة بها فقط ولا تشمل الأدوات المحضونة بتلك الأدوات المحضونة في الأداة نفسها. تخيل الأدوات التالية:

```
Form1
    GroupBox1
        TextBox1
        TextBox2
        TextBox3
    GroupBox2
        TextBox4
        TextBox5
```

الخاصية Form1.Controls ستعود بـ GroupBox1 وGroupBox2 فقط، والخاصية GroupBox1.Controls ستعود بصناديق النصوص الثلاثة الأولى، وبالمثل فإن صندوقي النص 4 و5 سيتم الحصول عليهما من الخاصية GroupBox2.Controls. [1] وهذا سبب استخدامنا للحلقات المتفرعة، إذ إن العملية تشابه التعامل مع ملفات ومجلدات ضمن مجلدات متفرعة.

والجدير بالذكر أن الأدوات المشتقة من الفئة ContainerControl لها الخاصية Controls، لذلك فـ Form وGroupBox وPanel وغيرها كلها أدوات مشتقة من هذه الفئة.

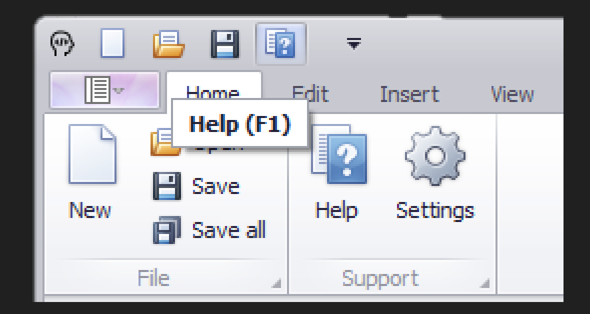
تخصيص اختصارات للأدوات والعناصر

للاختصارات دور كبير في زيادة سهولة تطبيقك، فهي تختصر على المستخدمين خطوات كثيرة للوصول لوظائف التطبيق. وكلما كانت الاختصارات مألوفة أكثر للمستخدمين كانت سهولة الاستخدام واعتياد المستخدمين على التطبيق واعتمادهم عليه أكثر. لذلك فخذ بعين الاعتبار تخصيص الاختصارات وفق المتعارف عليه، لا تجعل اختصار إنشاء ملف جديد Ctrl + Z مثلا! لا تجلطنا 😒. (هذا فضلًا عن أن تخصيص اختصار لأداة أو عنصر ما سيؤدي

[1] انظر "برمجة أطر عمل NET." لـ تركي العسيري (ص: 487-488).

لنتائج غير مدروسة في حال كان الاختصار مخصص لأداة أخرى! تخيل استخدام الاختصار Ctrl + C لفتح قائمة ما وهو مستخدم في صندوق نص للصق النصوص.. (🤦‍♂️)

يمكنك تخصيص الاختصارات من الخاصية ItemShortcut، اضبط اختصار الأمر Help – مثلًا – على F1. عند إيقاف مؤشر الفأرة بشكل مؤقت فوق الأمر الذي تم ضبط اختصار له سيظهر الاختصار بجانب اسم الأمر ضمن قائمة تلميح.



تخصيص تلميحات للأدوات والعناصر

يمكنك تخصيص تلميحات مختصرة لأدواتك عن طريق الخاصية Hint مما يجعلها أسهل على المستخدمين، لاحظ:



كما يمكنك تخصيص تلميحات مفصَّلة عن طريق الخاصية SuperTip:

SuperToolTip Editor

Properties

Title: img svg uri

New

Contents: img svg uri

Create a Task or Project file.

Show footer separator

Footer: img svg uri

Task files are individual file, whearas Project files are multi-Task files

Allow Html text: Default

Preview

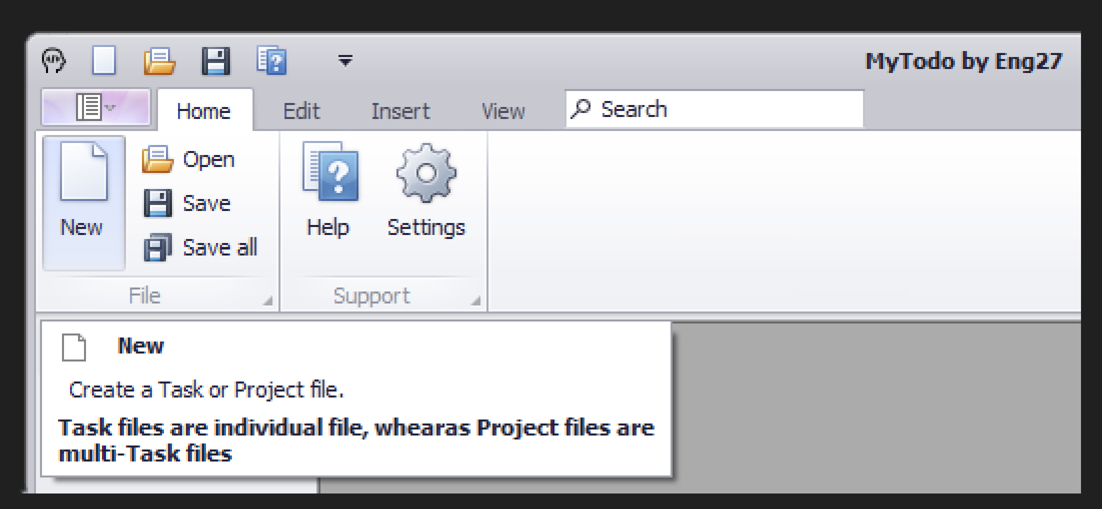
New

Create a Task or Project file.

Task files are individual file, whearas Project files multi-Task files

Version 19.1.6.0

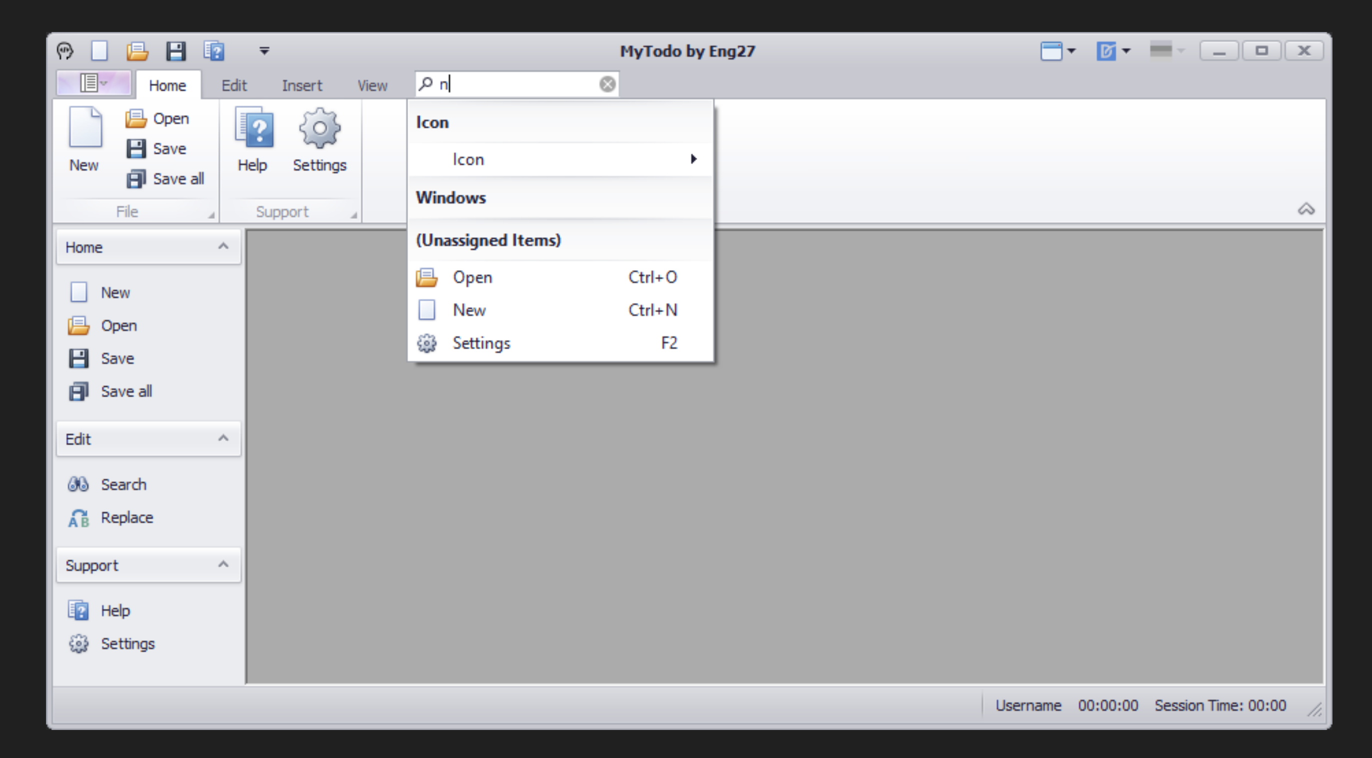
OK Cancel



عنصر البحث SearchItem

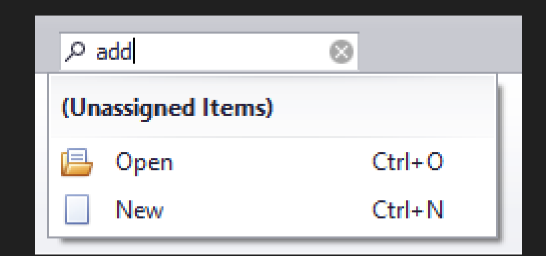
بإضافة إمكانية البحث عن عناصر ديف إكسبريس – وهي الأدوات الفرعية الموجودة داخل النافذة الرئيسية – فإنك ستجعل استخدام تطبيقك أسهل، خصوصًا إذا كانت أدواته وعناصره كثيرة!

يمكنك – كما وضحنا سابقًا – إضافة عنصر البحث لأداة الشريط RibbonControl من خلال تفعيل الخاصية ShowSearchItem، وبتفعيلها يمكن لمستخدم تطبيقك البحث عن الأدوات عوضًا عن البحث عنها في قوائم وأقسام التطبيق:



يمكنك إضافة كلمات مفتاحية للعناصر بحيث يمكن للمستخدم البحث عن العناصر سواءً أدخل اسمها أم أدخل الكلمات المفتاحية الخاصة بهذه العناصر، وذلك من خلال الخاصية SearchTags. هذه الخاصية تأخذ قائمة من الكلمات المفتاحية على شكل قيمة نصية يفصل بين الكلمات المفتاحية فيها بفاصلة comma، كما يمكن لأكثر من أداة أن تتشارك كلمة مفتاحية أو أكثر.

اضبط خاصية SearchTags لبعض العناصر على قيم مختلفة عن اسمها وتؤدي نفس المعنى، مثل "add, insert, create" للعنصر bbtnNew و"add, insert" للعنصر bbtnOpen، وغيرها من العناصر، ثم ابحث عن كلمة معينة ولتكن "add":



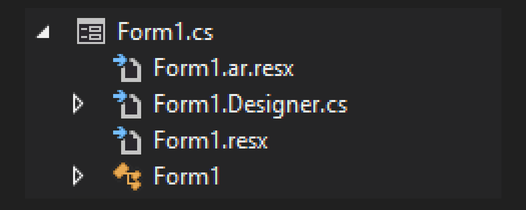
تعدد اللغات Localizable

توفّر لك منصة دوت نت إمكانية ترجمة تطبيقك لأكثر من لغة، وذلك من خلال الخاصية Localizable. كما يمكنك ضبط لغات التطبيق من خلال الخاصية Language. هذه الخاصية يجب ضبطها لكل نافذة على حدة ضمن مشروعك.

إضافة اللغات لنوافذ التطبيق بسيط للغاية، غيّر اللغة عبر الخاصية Language ثم اضبط عناوين وقيم نصوص أدوات النافذة على الترجمة المطلوبة. كرر ذلك للغات الثانية. عند تغيير قيمة الخاصية Language للغة تم ضبطها فستتغير عناوين وقيم نصوص الأدوات للترجمة المضبوطة.

لنبدأ باللغة العربية.. اضبط الخاصية Localizable للنافذة Form1 على True، ثم الخاصية Language على Arabic. ثم غير جميع عناوين Caption ونصوص Text الأدوات للترجمة المطلوبة. ولا تنسَ بدايةً تغيير الخاصيتين RightToLeft وRightToLeftLayout لـ Yes وTrue بالترتيب، فهذا من باب أولى!!

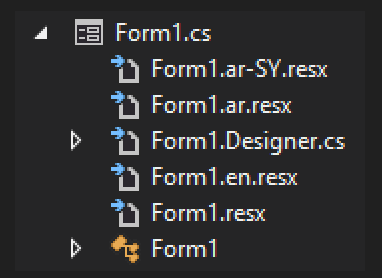
بعد ضبط أدوات النافذة لاحظ أن النافذة بات فيها ملف جديد وهو Form1.ar.resx:



أضف اللغة الإنكليزية للغات النافذة، حتى نتمكن من التحويل إليها. عند إضافة اللغة الإنكليزية سيتم إرجاع النافذة للغة الافتراضية – المحفوظة في Form1.resx – مما يجعل تفاصيل النافذة باللغة الإنكليزية نفسها باللغة الافتراضية، لذلك قم بتغيير خاصية من خصائص أدوات النافذة حتى يحدث تغيير (يمكنك حذف التغيير لاحقاً).

أضف لهجة عربية، السورية مثلًا (أو أي لهجة ترغب بها)، واضبط خصائص النافذة وفق اللهجة التي اخترتها.

بات لديك:



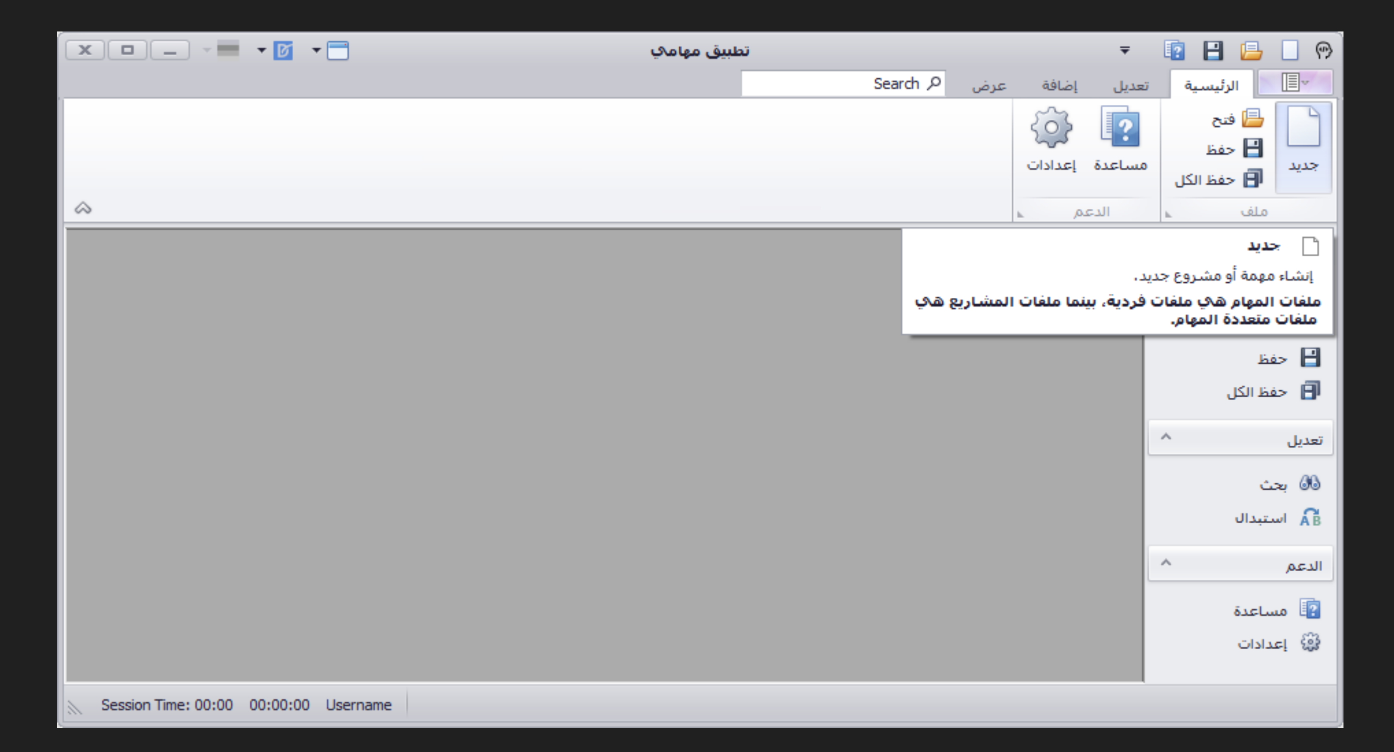
عملية تغيير لغة النموذج تكون عند استنساخه، لذلك فعلى المستخدم إعادة تشغيل التطبيق عند تغيير اللغة (في الواقع أكوادك هي من ستعيد تشغيل التطبيق، المستخدم عليه حفظ ملفاته المفتوحة فقط). ولتغيير اللغة ستحتاج مجالات الأسماء: System.Threading وSystem.Globalization، والكود التالي قبل إجراء تهيئة مكونات النموذج:

```
public Form1()
{
    string lang = "ar"; // هذه القيمة تؤخذ من إعدادات التطبيق
    Thread.CurrentThread.CurrentUICulture = new CultureInfo(lang);
    InitializeComponent();

    //...
    //Codes
}
```

المتغير lang يأخذ قيمته من ملف – أو قاعدة بيانات – يمثل إعدادات البرنامج، بحيث يغير المستخدم لغة البرنامج من قسم الإعدادات فيُحفظ خيار اللغة الذي اختاره المستخدم ويعاد تشغيل البرنامج لتحميل اللغة الجديدة. قيمة هذا المتغير متعلقة باللغة التي

اختارها المستخدم، والتي يمكنك معرفة قيمتها النصية من مصادر النموذج، فاللغة العربية السورية هي باسم Form1.ar-SY.resx، وعليه فالمتغير النصي lang يأخذ القيمة "ar-SY".



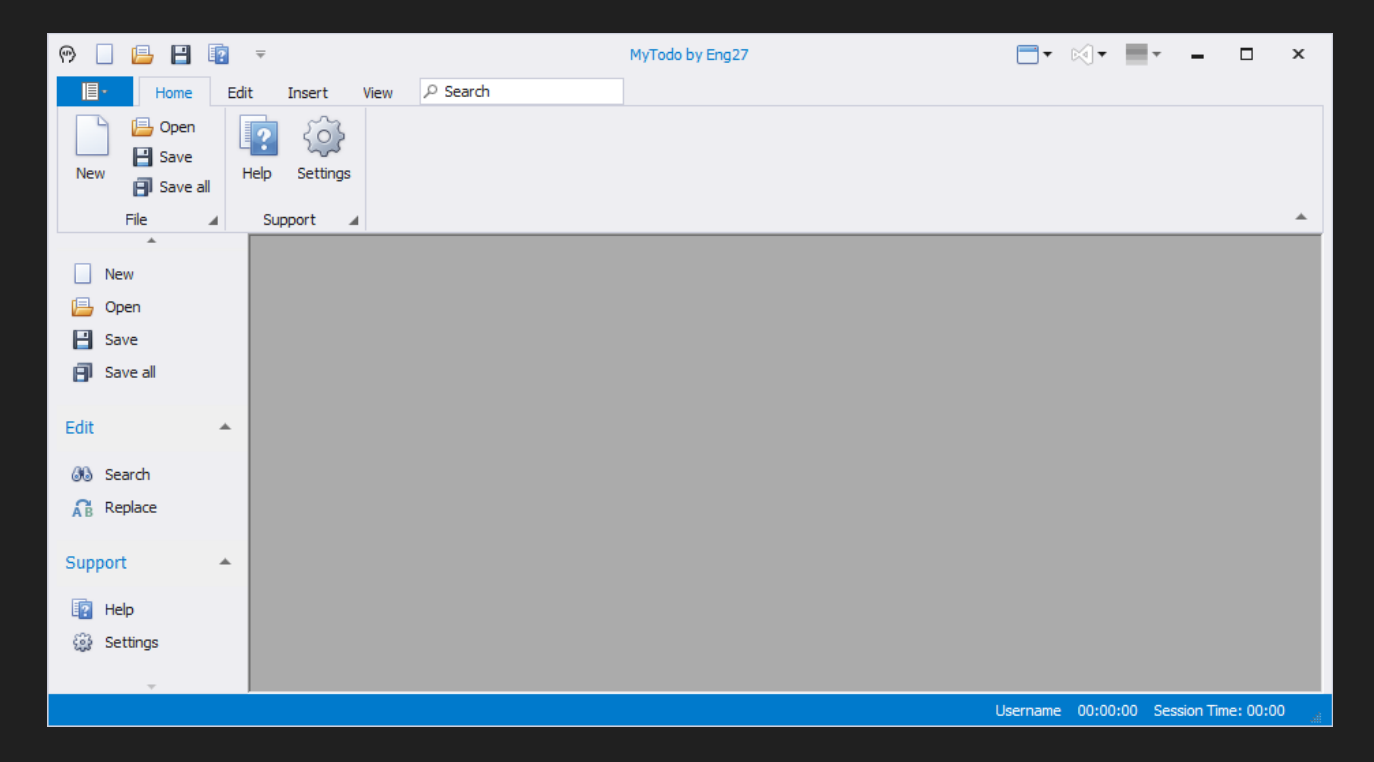
تغيير سمة Theme التطبيق

يمكن لتطبيقات ديف إكسبريس أن تكون بأكثر من سمة Theme (أو Skin)، يمكن للمستخدم تغيير السمة من خلال الأدوات التي أضفناها في شريط العنوان (موجودة على يمين الصورة التالية)، ولكن عليك ضبط السمة التي اختارها المستخدم في كل مرة يتم تشغيل تطبيقك فيها.. وذلك من خلال إضافة الأداة DefaultLookAndFeel واستخدام الكود:

```
public Form1()
{
    InitializeComponent();
    defaultLookAndFeel1.LookAndFeel.SkinName = "Visual Studio 2013 Light"; // مثلا

    //...
    //Codes
}
```

لاحظ:



تفضيلات المستخدم واختياراته

يمكن للمستخدم تخصيص كثير من الأمور في التطبيق، مثل عناصر شريط الوصول السريع والسمة Theme الحالية ولغة البرنامج وغيرها الكثير. وهذه الخيارات يجب أن تحفظ في مكان ما ويؤخذ بها في مكانها المناسب. يمكنك حفظها في ملف ما أو في الرجستري أو في قاعدة بيانات، أو حتى في مصادر المشروع!

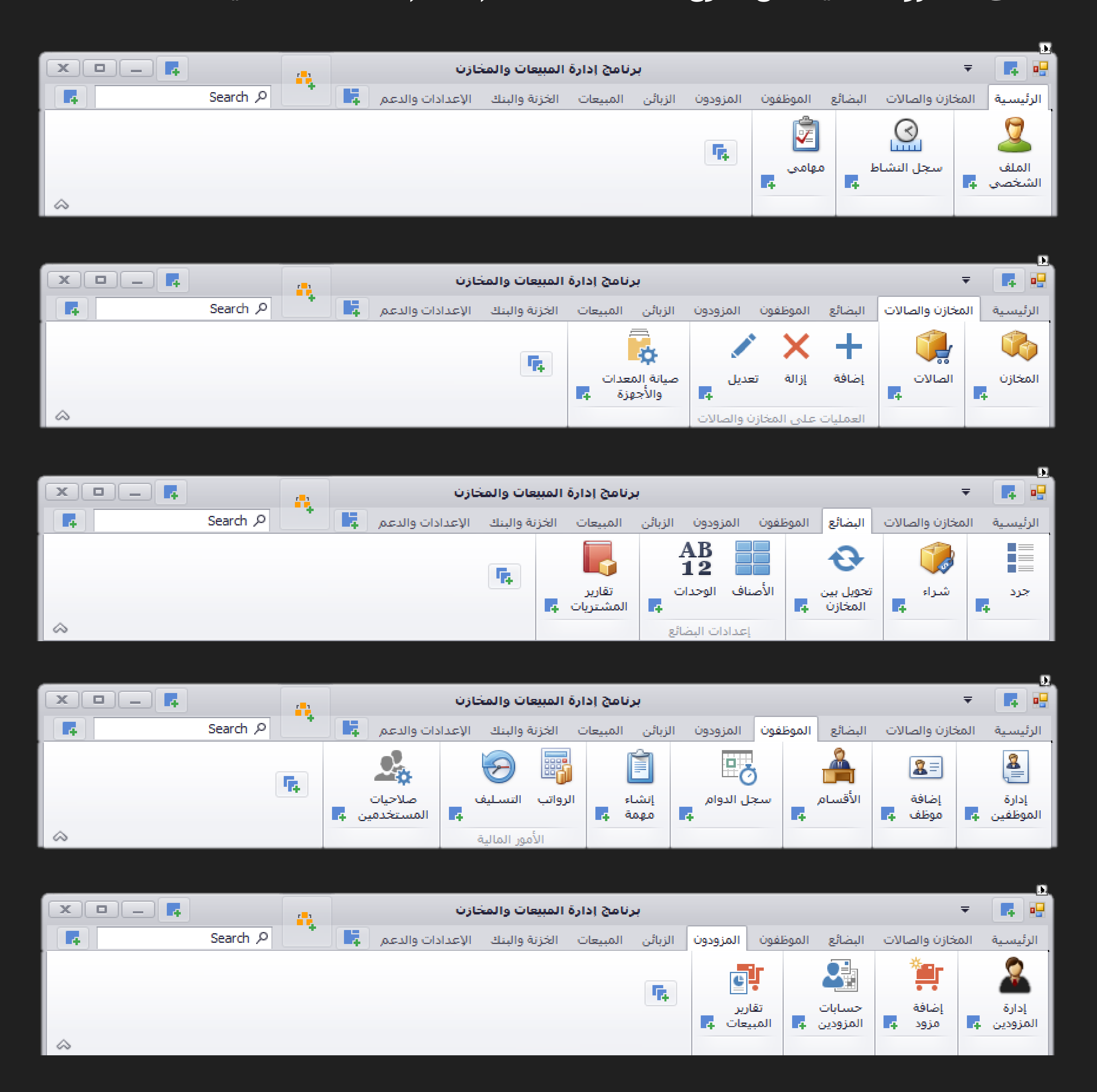
المشروع الثاني، مدير متجر مبيعات وتخزين

يشترك هذا المشروع مع المشروع السابق بنافذة الشرائط RibbonForm ويختلف عنه في أن النافذة الرئيسية لهذا المشروع ليست نافذةً أم MDI، أي أنها لن تكون متعددة المستندات. فأزرار هذه النافذة تنقلك لنوافذ أخرى، لكل نافذة مهمة معينة.

كما لم تكن الغاية من المشروع الأول كيفية برمجة وتصميم برنامج إدارة مهام ومشاريع – وإنما كيفية استخدام أدوات ديف إكسبريس – فالمشروع الثاني أيضًا ليست الغاية منه كيفية برمجة وتصميم مدير متجر مبيعات وتخزين، ولو أن المشروعين يحويان في

طياتهما بعض الأمور التطبيقية، فهذا الفصل معني بتقديم أمثلة على التصميم وليس البرمجة ولا حتى طرق التصميم. فعندما تأخذ هذا التوضيح بعين الاعتبار فإنك ستحصل على الغاية المرجوة من الفصل بشكل أفضل ولن تقول "ما هذا الشرح الناقص!" هذا فضلًا عن أنك ستتمكن من إنشاء مشاريع أخرى بنفس مبدأ هذه المشاريع مع قناعةٍ بالمحتوى ورضىً عنه.

أنشئ مشروعًا جديدًا من النوع RibbonForm ثم صمم الصفحات التالية:





يفضل عنونة المجموعات التي تحوي أكثر من عنصر، كما يمكنك تغيير حجوم عناصر المجموعات تبعًا لأهميتها مما يجعل أداة الشريط لديك أسهل للمستخدم على المدى البعيد.

اضبط الخاصية ShowCaptionButton على false وText على " " وذلك لجميع مجموعات الصفحات الموضحة بالصور السابقة. وفعّل الخاصية ShowSearchItem للشريط.

وبالنسبة للنافذة Form1 غير قيمتها النصية لما هو موضح في الصور السابقة واجعل اتجاه النافذة نحو اليمين وحالة النافذة WindowState على Maximized.

يمكنك إضافة بعض الأوامر الأخرى التي ترى أهميتها، ويفضل إضافة بعض العناصر – كما في المشروع السابق – لشريط الوصول السريع وشريط العنوان (مثل أدوات تغيير مظهر التطبيق وغيرها).

فكرة المشروع:

- الصفحة الرئيسية تعطي المستخدم إمكانية الوصول لبياناته الشخصية وسجل نشاطه – ونشاطات من هم ضمن قسمه – ومهامه (يمكن للمستخدم أن يضيف مهامه كما يمكن لمشرف قسمه ذلك).
- صفحة المخازن والصالات فيها أماكن الشركة وتفاصيل آلاتها وأجهزتها، بالإضافة للعمليات على هذه الأماكن كإضافة مخازن أو صالات أو أجهزة أو آلات.
- صفحة البضائع – أو صفحة المشتريات – تحوي كل ما دخل للشركة، مع إمكانية التحويل بين أماكن الشركة (من مخزن لمخزن أو من صالة لصالة أو من مخزن لصالة أو العكس)، كما يتم فيها ضبط الأصناف والوحدات [1].
- صفحة شؤون الموظفين تعنى بضبط كل ما هو متعلق بموظفي الشركة، كإضافة الموظفين وإدارتهم والتعاملات المالية معهم. كما يمكن ضبط المسميات الوظيفية من خلال الأمر "الأقسام" أو إنشاء المهام للموظفين أو تسجيل حضورهم من هذه الصفحة. بالإضافة إلى أنه من خلال هذه النافذة يتم ضبط صلاحية كل موظف يستخدم هذا البرنامج، فبعض الموظفين يمكنهم الوصول للأماكن فقط، وغيرهم للمزودين والعملاء، وموظفين آخرين وظيفتهم المبيعات فقط، كما يمكن لنفس الموظف أن يصل لأكثر من صفحة، وهذا كله يتم ضبطه من الأمر "صلاحيات المستخدمين"، للمزيد أحيلك لدورة الأستاذ أيمن سلطان [صلاحيات المستخدمين C# & SQL Server] [2] على اليوتيوب.

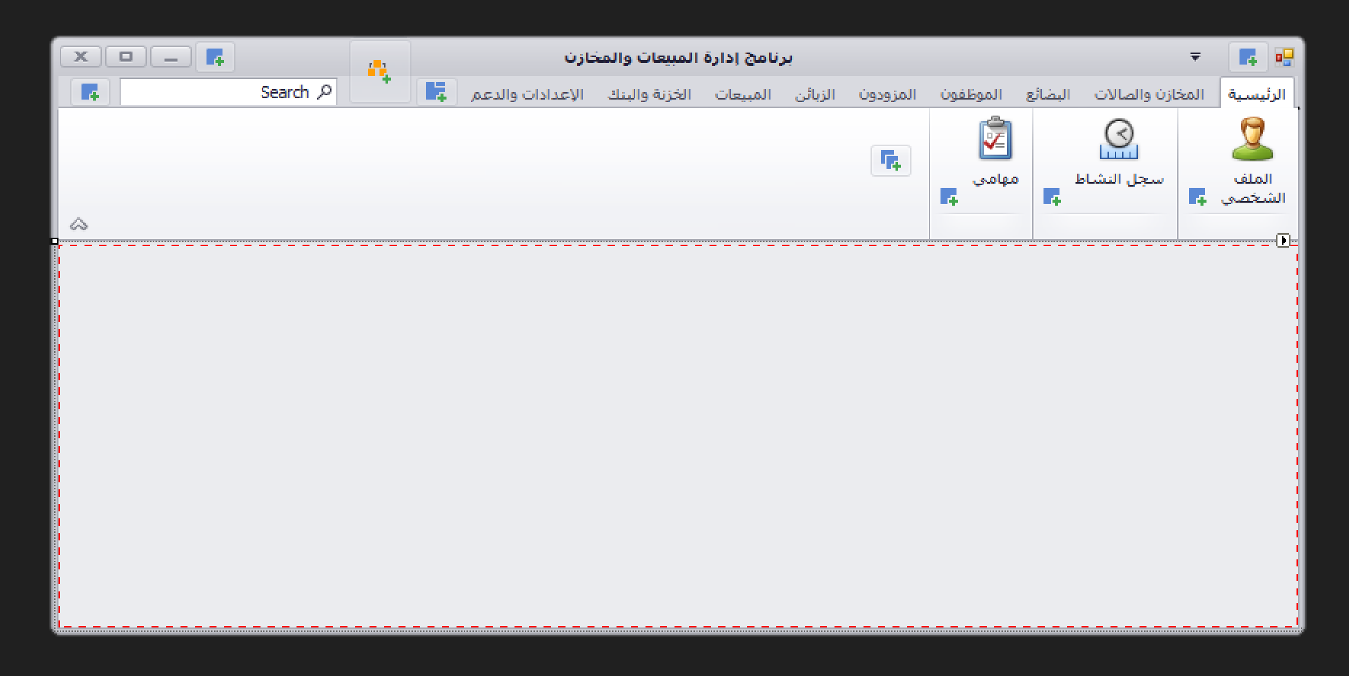
---

[1] الصنف: نوع المادة، كالمواد الغذائية والمواد المنزلية وغيرها. الوحدة: وحدة القياس، كاللتر والكيلوغرام والغرام والكرتونة والقطعة وغيرها.
[2] دورة صلاحيات المستخدمين للأستاذ أيمن سلطان https://www.youtube.com/playlist?list=PLmKNfJisB1cyL0tpOJq2ck5NL9MqUJpYN

- صفحتا المزودين – أو الموردين – والزبائن – أو العملاء – يمكن من خلالهما ضبط الجهات التي تتعامل معها الشركة.
- صفحة المبيعات هي الصفحة المشهورة في برامج المحاسبة، وهي ما يتعامل معه المحاسب "الكاشير" مباشرة.
- صفحة البنك والخزنة فيها مصادر مال الشركة.
- صفحة الإعدادات والدعم فيها يتم ضبط إعدادات البرنامج العامة – بعضها يخص المستخدم وبعضها يخص البرنامج، والوصول للإعدادات مقيد بصلاحيات كل مستخدم – والحصول على المساعدة وتفاصيل عن البرنامج.

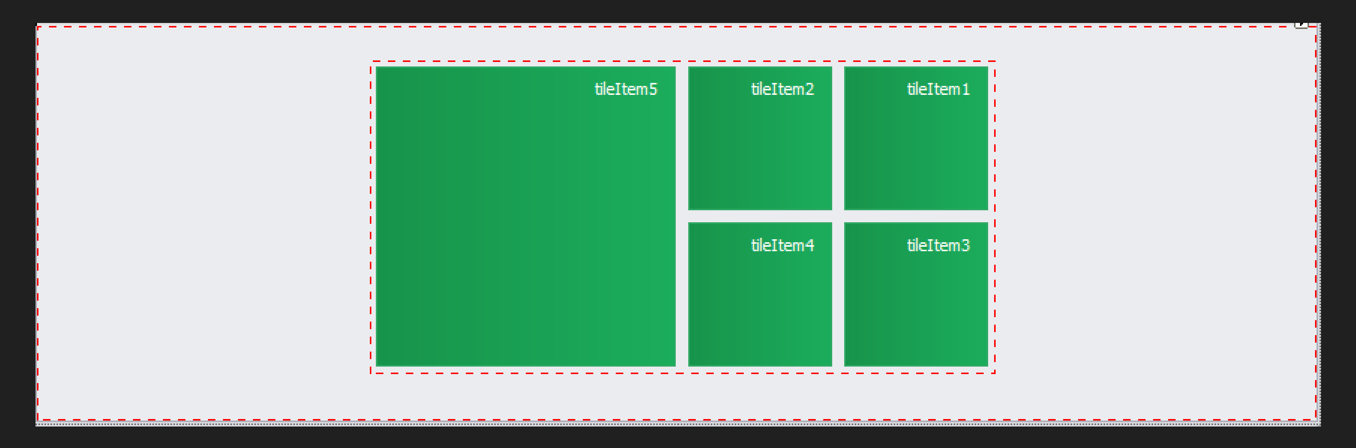
الأداة TileControl

تميز نظام ويندوز 8 بوجود الأزرار المسطحة أو MetroButtons في قوائمه، والتي يوجد فيها حركة Animation لإضفاء روح على القوائم، كما ورث نظام ويندوز 10 هذا. يمكنك إنشاء قائمة من هذا النوع باستخدام أداة TileControl. من صندوق الأدوات اسحب هذه الأداة وألقها على نافذة برنامجك، وغير خاصية Dock فيها لـ Fill.

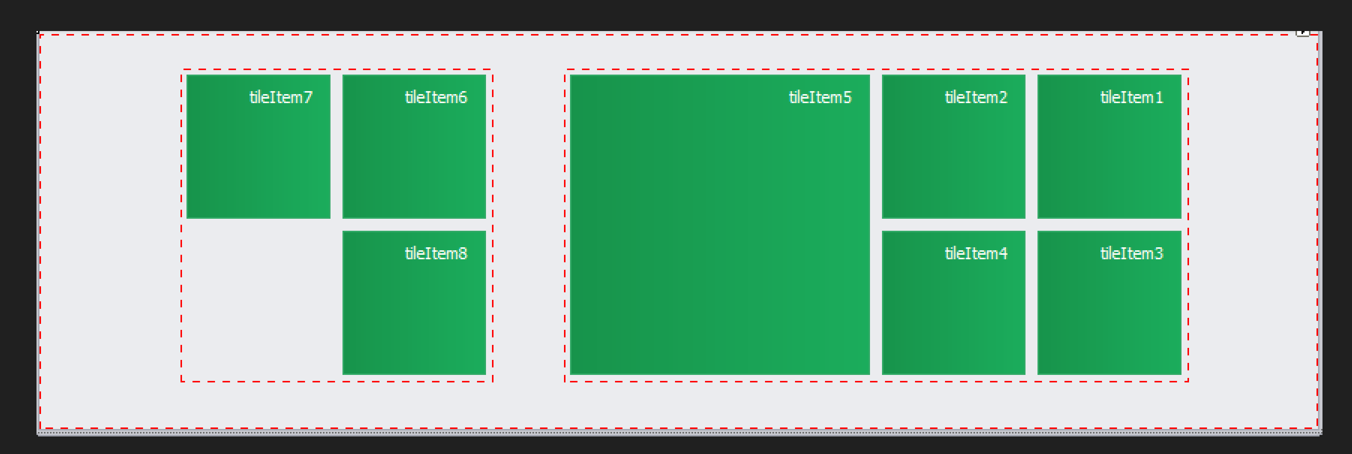


من اختصار الأوامر المميزة SmartTag لأداة tileControl1 – أو بالنقر بالزر الأيمن عليها – اختر Add Group لإضافة مجموعة:

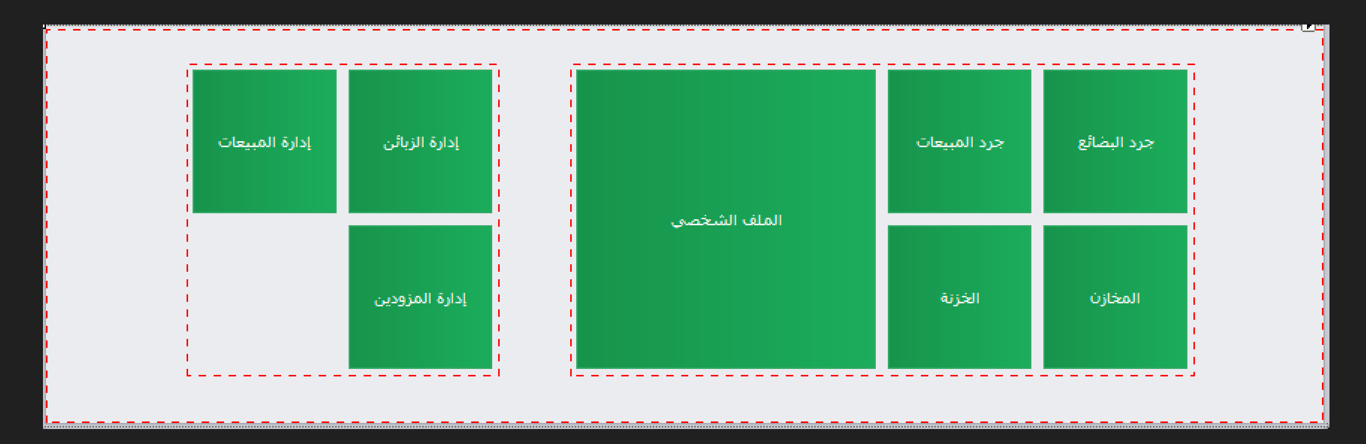
عدل حجم العنصر ItemSize لـ 200 من خصائص tileControl1 ثم قم بإضافة أربعة عناصر Small Item وعنصرًا متوسطًا Medium Item للمجموعة (وذلك بالنقر بالزر الأيمن على المجموعة):



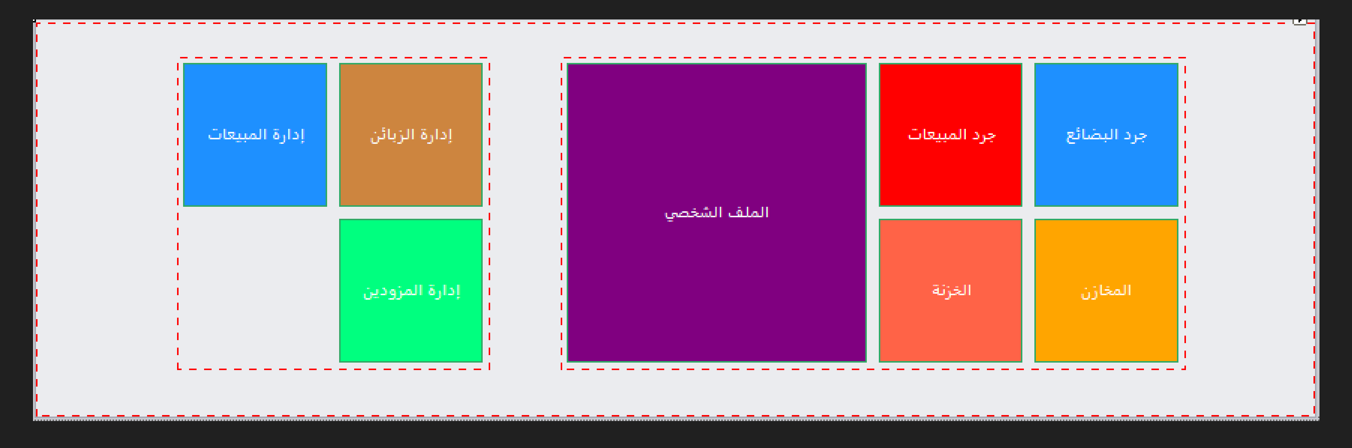
أضف مجموعة أخرى فيها ثلاثة عناصر صغيرة:



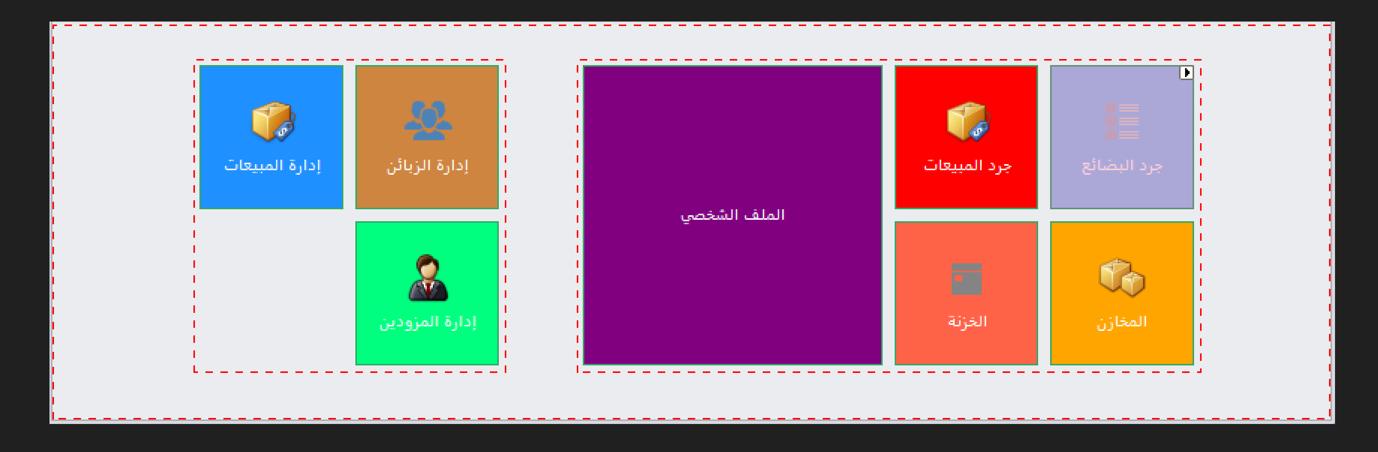
غير الخاصية TextAlignment لجميع العناصر للقيمة MiddleCenter وغير قيمها النصية كما هو موضح:



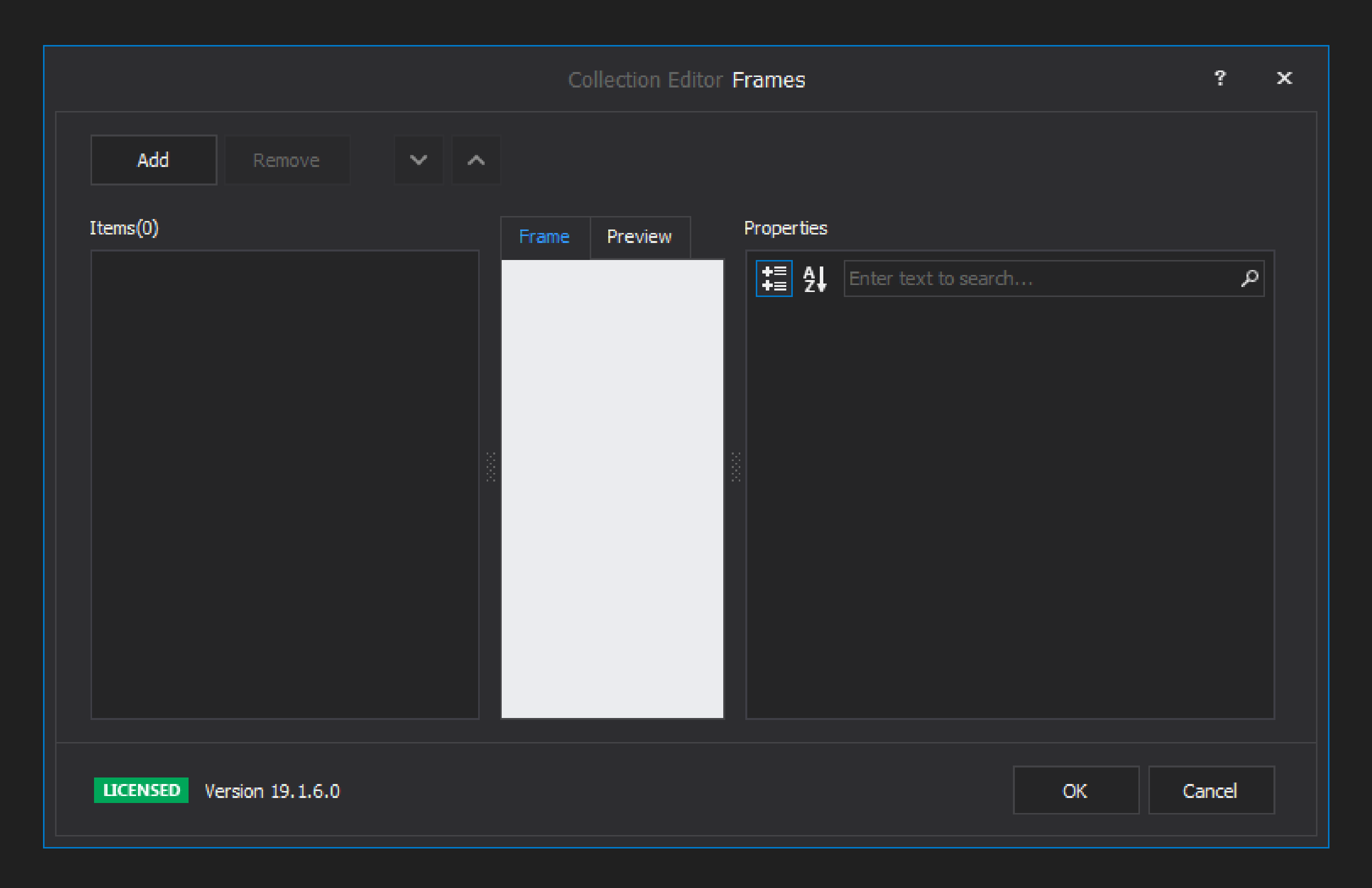
غير ألوان وخطوط العناصر:



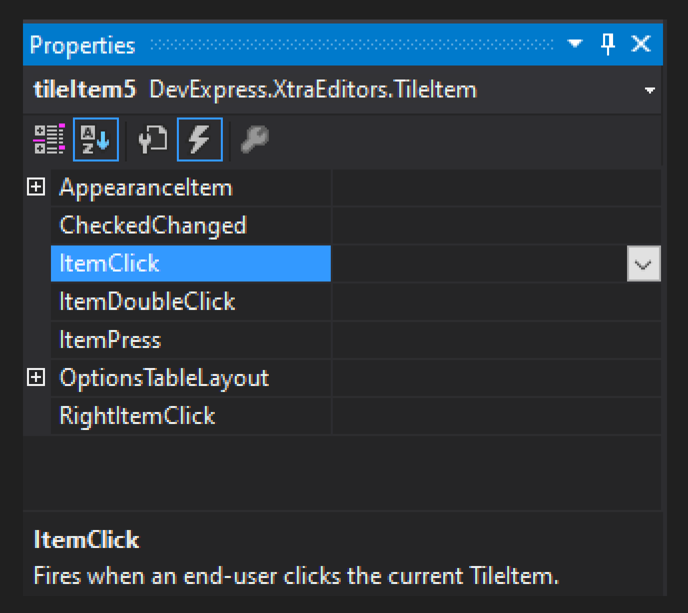
غير الخاصية ImageAlignment لـ MiddleCenter والخاصية ImageToTextAlignment لـ Top ثم اضبط بعض الصور:



ولإضافة حركة Animation للأزرار انقر بالزر الأيمن على العنصر المراد ضبط الحركة له ثم Edit Animation Frames [1]:



انقر على Add لإضافة مقطع واضبط لونه على Purple ثم مقطعًا آخر واضبط لونه على Fuchsia (جرب عدل بقية الخصائص وأضف مقاطع أخرى). كما يمكنك تغيير الفترة بين كل مقطع وآخر من خلال الخاصية FrameAnimationInterval.



يمكنك من خلال عناصر أداة TileControl التحكم بالأحداث التالية:

[1] مبدأ الحركة هنا بسيط للغاية، يتم ضبط عدة مقاطع Frames بحيث يتم الانتقال بين المقطع والآخر خلال فترة زمنية معينة. المقاطع ماهي إلا نسخ من العنصر نفسه بخصائص مختلفة (أي أنك عندما تنشئ مقاطع جديدة فإنك تنشئ نفس الأداة بتفاصيل مختلفة).

ما يهمنا منها بالدرجة الأولى – في مشروعنا هذا على أقل تقدير – حدث ItemClick، والذي سيؤدي عند تفجيره إلى القيام بالوظائف المتعلقة بهذه الأزرار، والتي يمكن الوصول إليها أيضًا من خلال عناصر الشريط العلوي.

نافذة تسجيل الدخول

من مستعرض المشروع Solution Explorer انقر بالزر الأيمن على اسم مشروعك ثم Add DevExpress Item ثم New Item، اختر نافذة من النوع XtraForm وسمها frmLogin، ثم صممها بالشكل التالي:



القيمة	الخاصية	الأداة
White	BackColor	frmLogin
Glow	FormBorderEffect	
None	FormBorderStyle	
False	MaximizeBox	
False	MinimizeBox	
False	ShowIcon	

	ShowInTaskbar	False
	StartPosition	CenterParent
textEdit1	EnterMoveNextControl	True
	HAlignment	Center
textEdit2	UseSystemPasswordChar	True
	HAlignment	Center
simpleButton1	BackColor	LightGoldenrodYellow
	Text	دخول
simpleButton2	Location [1]	MiddleCenter

عند تشغيل البرنامج تُلغى جميع صفحات الشريط وعناصر أداة tileControl1 وتظهر النافذة frmLogin مباشرةً، فإذا لم يستطع المستخدم تسجيل الدخول يتم إيقاف تشغيل البرنامج. استخدم هذا الكود – وطوره – للتحكم بتسجيل الدخول:



```
using System;
using DevExpress.XtraEditors;

namespace DevExpressTest2
{
    public partial class frmLogin : XtraForm
    {
        int id;
        string password;

        public frmLogin()
        {
            InitializeComponent();
        }

        private void simpleButton1_Click(object sender, EventArgs e)
        {
            if (SearchFor(textEdit1.Text)) // إذا كانت نتيجة التابع إيجابية يتم تسجيل الدخول
            {
                Close();
                this.DialogResult = System.Windows.Forms.DialogResult.OK; // ملاحظة1
            }
            else
                XtraMessageBox.Show("اسم المستخدم أو كلمة المرور خاطئة", "خطأ في تسجيل الدخول");
        }
```

[1] تجد هذه الخاصية ضمن ImageOptions.

```
        bool SearchFor(string username)
        {
            id = SearchForIdByUserName(username); // ملاحظة2
            if (id > 0) // ملاحظة3
            {
                if (textEdit2.Text == GetPasswordByID(id)) // ملاحظة4
                {
                    Form1.userId = id; // ملاحظة5
                    return true;
                }
            }
            else
                return false;
        }

        int SearchForIdByUserName(string username)
        {
            // يتم البحث في قاعدة البيانات عن اسم المستخدم الذي تم إدخاله .. فإذا كان موجودًا يتم إرجاع قيمة معرّفه
            // وإلا يتم إرجاع القيمة 0
        }

        string GetPasswordByID(int Id)
        {
            // يتم جلب كلمة السر بعد معرفة قيمة معرّفه
            return password;
        }

        private void simpleButton2_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
    }
}
```

وفي النافذة الأساسية:



```
using System;
using System.Windows.Forms;
using DevExpress.XtraBars.Ribbon;

namespace DevExpressTest2
{
    public partial class Form1 : RibbonForm
    {
        public static int userId; // ملاحظة6
        string userName;

        public Form1()
        {
            InitializeComponent();
            DisableControls();
        }
```

```
        private void Form1_Load(object sender, EventArgs e)
        {
            frmLogin fl = new frmLogin();
            DialogResult dr = fl.ShowDialog(); // ملاحظة7
            if (dr == System.Windows.Forms.DialogResult.OK)
                LoginSucceeded();
            else
                DisableControls();
        }

        void LoginSucceeded()
        {
            ribbonControl1.Enabled = true;
            tileControl1.Enabled = true;
            GetUserName(); // ملاحظة8
            DisableControlsUserDontHavePermissions();
        }

        void DisableControls()
        {
            ribbonControl1.Enabled = false;
            tileControl1.Enabled = false;
        }

        void GetUserName() {
            // يتم الاعتماد على معرّف المستخدم لتحديد اسمه
            // حصلنا على المعرف من نافذة تسجيل الدخول
            userName = GetUserNameFromDB(id);
        }

        void DisableControlsUserDontHavePermissions() {
            // اعتمادًا على معرّف المستخدم نحصل على ما لا يمكن للمستخدم الوصول إليه
            // فنقوم بإلغاء إظهار أو تمكين الصفحات والعناصر التي لا يمكنه الوصول إليها
        }
    }
}
```

الملاحظات:

1- إذا كان اسم المستخدم وكلمة المرور صحيحين يتم جعل قيمة النافذة هي القيمة OK، وبالتالي نمكّن المستخدم من الأزرار والعناصر والصفحات التي له صلاحية الوصول إليها.

2- إذا كان اسم المستخدم موجودًا فإن التابع SearchForIdByUserName يعيده كقيمة له، وإلا فإنه يعيد القيمة 0.

3- نبحث عن معرّف المستخدم وذلك اعتمادًا على اسم المستخدم الذي تم إدخاله.

4- إذا كان المعرف id أكبر من 0 فإن هذا يعني أن المستخدم موجود، وعندها يتم التحقق من كلمة السر التي أدخلها المستخدم وذلك بمقارنتها مع كلمة السر

التي يتم جلبها من قاعدة البيانات من خلال التابع GetPasswordByID اعتماداً على معرّف المستخدم.

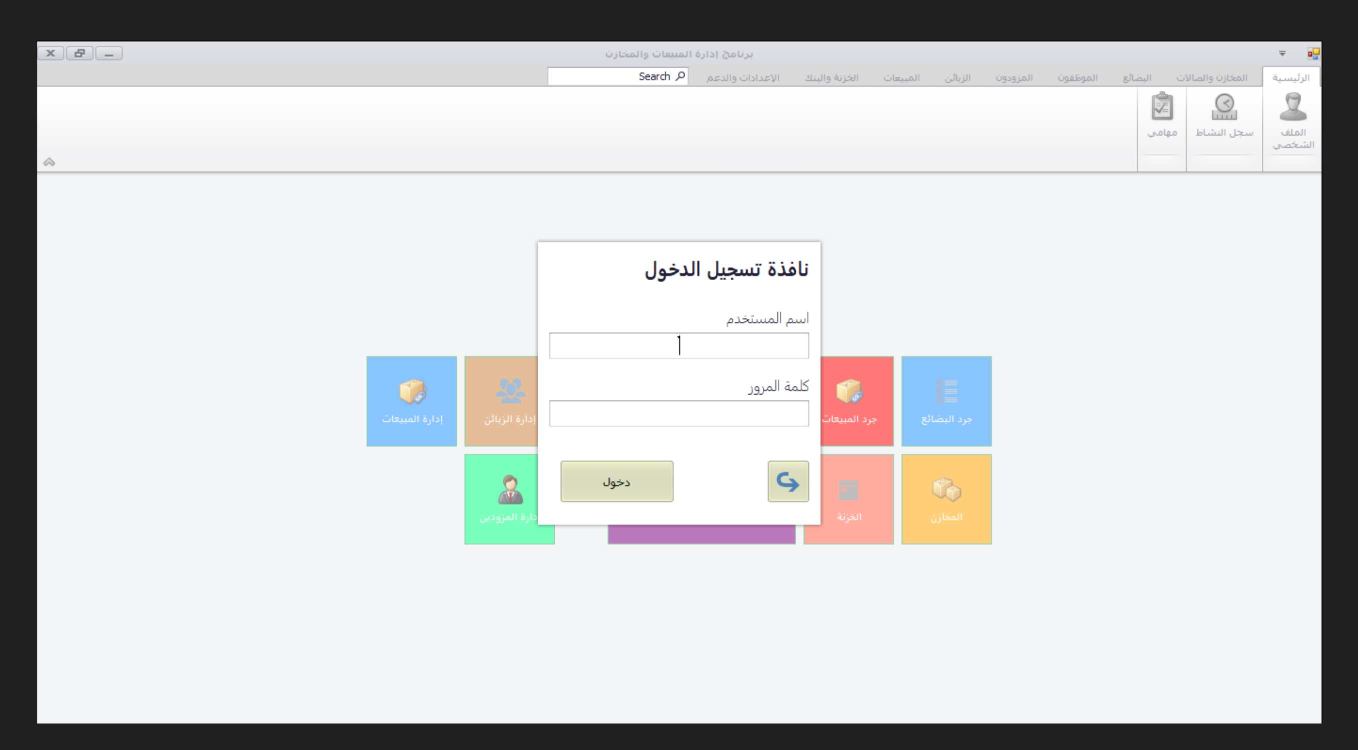
5- إذا كانت كلمة السر التي أدخلها المستخدم صحيحة، يتم جعل قيمة المتغير userId – الموجود في النافذة الرئيسية – مساوية لقيمة معرّف المستخدم الذي تم الحصول عليه سابقًا.

6- المتغير الستاتيكي يمكن الوصول إليه مباشرة دون استنساخ كائن من الفئة الموجود فيها هذا المتغير (راجع السطر المشار إليه بالملاحظة5).

7- عند تعريف كائنات من نوع الخصائص والطرق التي يتم استدعاءها فإن الكود سيكون أبسط، لأنه يمكنك تجزئة الكود إلى أجزاء يمكن لاحقًا تحليلها ومعالجتها بشكل أسهل.

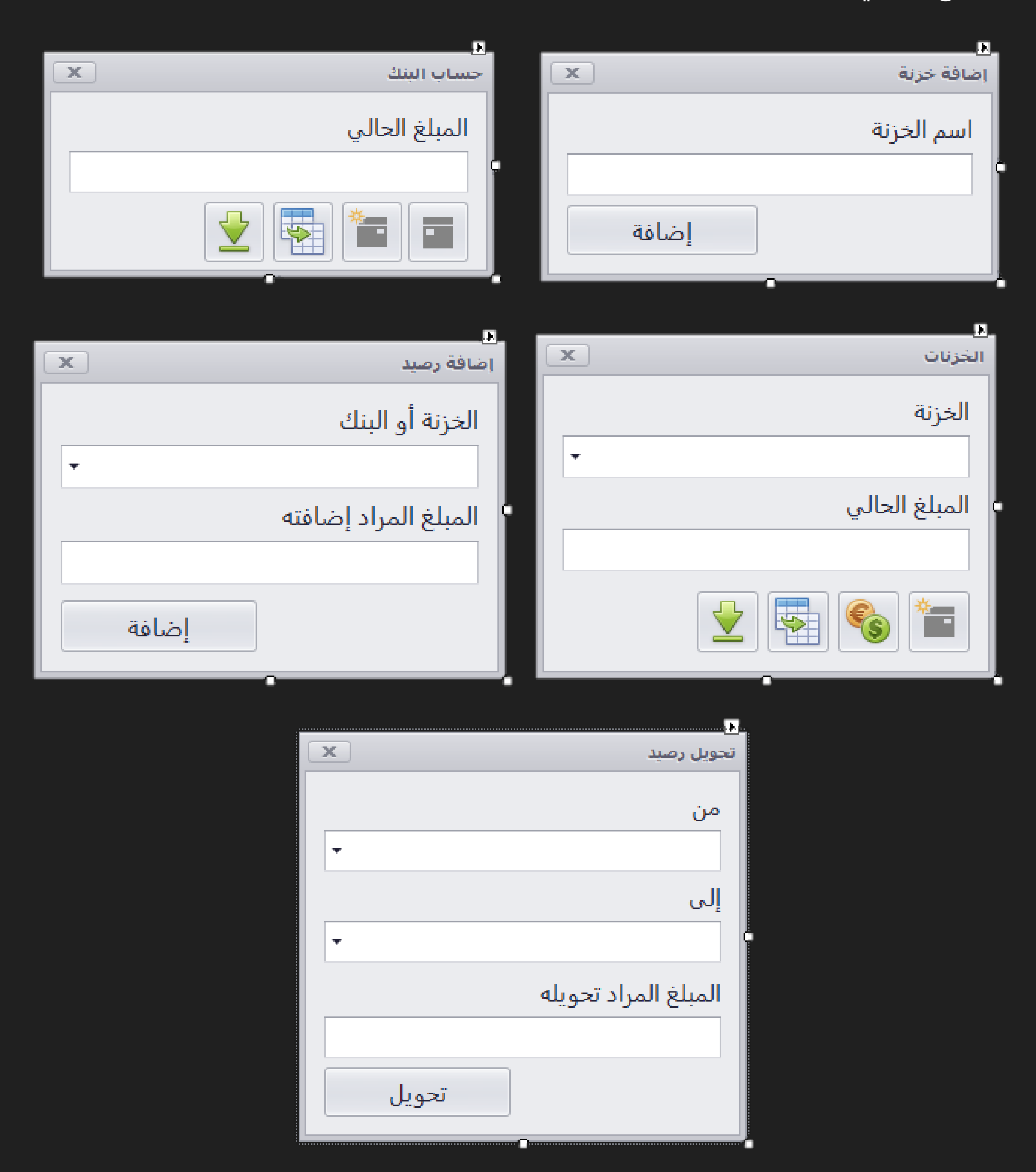
8- على اعتبار أن معرّف المستخدم الذي تم تسجيل الدخول من خلاله معنا، يمكننا الحصول على جميع تفاصيله متى أردنا، وكمثال: الحصول على اسم المستخدم.

عند تشغيل البرنامج سيتم فتح نافذة البرنامج وكل مافيها غير مفعَّل وفي وسطها نافذة تسجيل الدخول كما يلي:



نوافذ الخزنة والبنك

أول ما سيقوم به مستخدم برنامجك – والذي يعتبر مدير المتجر – هو ضبط ميزانيته وخزناته، لذلك سنصممها أولًا.. أضف مجموعة من النوافذ من النوع XtraForm وصممها بالشكل التالي:



نوافذ العمليات على الخزنات والبنك (الإضافة والطرح والتحويل) سيتم الخروج منها عند القيام بالمهمة المسندة لها (النقر على زر "إضافة"، "تحويل"، ...)، بينما نوافذ عرض البيانات فلا مهمة لها سوى عرض البيانات. وعليه فإنه من المفيد إضافة أزرار تنقل المستخدم لنوافذ ذات صلة كما في نافذتي "حساب البنك" و"الخزنات".



حاول جعل نوافذ برنامجك التي لها نفس الغرض متشابهة مما يجعل استخدامه أسهل وأبسط على المستخدم.

اعتمد على الجدول التالي لضبط خصائص الأدوات:

الأداة	الخاصية	القيمة
XtraForm [1]	AutoScaleMode	None
	Font	Segoe UI, 12pt
	MaximizeBox	False
	MinimizeBox	False
	RightToLeft	Yes
	RightToLeftLayout	True
	ShowIcon	False
	ShowInTaskbar	False
	StartPosition	CenterParent
frmAddMoney	MaximumSize	300, 215
	MinimumSize	300, 215
	Size	300, 215
	Text	إضافة رصيد
frmAddTresury	MaximumSize	300, 150

[1] اضبط جميع النوافذ على هذه الخصائص.

	MinimumSize	300, 150
	Size	300, 150
	Text	إضافة خزنة
frmBankAccount	MaximumSize	300, 150
	MinimumSize	300, 150
	Size	300, 150
	Text	حساب البنك
frmExchangeMoney	MaximumSize	300, 275
	MinimumSize	300, 275
	Size	300, 275
	Text	تحويل رصيد
frmTresury	MaximumSize	300, 225
	MinimumSize	300, 225
	Size	300, 225
	Text	الخزنات
comboBoxEdit [1]	TextEditStyle	DisableTextEditor
textEdit [2]	EnterMoveNextControl	True
	Font	Segoe UI, 12pt
	HAlignment	Center
simpleButton	Size	40, 40
للنوافذ ذات الصلة	Text	

* اضبط جميع الخطوط على Segoe UI, 12pt.

[1] اضبط جميع صناديق اللوائح المنبثقة على هذه الخاصية.
[2] اضبط جميع صناديق النصوص على هذه الخصائص.

استخدم الأكواد التالية:



```
using System;
using System.Collections.Generic;
using DevExpress.XtraEditors;

namespace DevExpressTest2
{
    public partial class frmAddMoney : XtraForm
    {
        public frmAddMoney()
        {
            InitializeComponent();
        }

        List<string> tresuries = new List<string>();

        private void frmAddMoney_Load(object sender, EventArgs e)
        {
            tresuries = GetTresuriesNames();
            comboBoxEdit1.Properties.Items.AddRange(tresuries);
            comboBoxEdit1.SelectedIndex = 0;
        }

        List<string> GetTresuriesNames()
        {
            // تحميل أسماء الخزنات من قاعدة البيانات .. وإعادتها كلائحة
            List<string> t = new List<string>() { "الخزينة الافتراضية",
                "الخزينة1",
                "الخزينة2" };
            t.Add("البنك");
            return t;
        }

        private void comboBoxEdit1_SelectedIndexChangd(object sender, EventArgs e)
        {
            // يتم البحث في قاعدة البيانات عن الخزنة المحددة وعرض تفاصيلها
        }
    }
}
```

```
using System;
using DevExpress.XtraEditors;

namespace DevExpressTest2
{
    public partial class frmBankAccount : XtraForm
    {
        public frmBankAccount() {
            InitializeComponent();
        }

        private void frmBankAccount_Load(object sender, EventArgs e)
        {
            textEdit1.Text = GetBankAccount();
        }
```

```
        private void simpleButton1_Click(object sender, EventArgs e)
        {
            frmTresury ft = new frmTresury();
            ft.ShowDialog();
        }

        private void simpleButton2_Click(object sender, EventArgs e)
        {
            frmAddTresury fat = new frmAddTresury();
            fat.ShowDialog();
        }

        private void simpleButton3_Click(object sender, EventArgs e)
        {
            frmExchangeMoney fe = new frmExchangeMoney();
            fe.ShowDialog();
        }

        private void simpleButton4_Click(object sender, EventArgs e)
        {
            frmAddMoney fam = new frmAddMoney();
            fam.ShowDialog();
        }

        string GetBankAccount()
        {
            // يتم البحث في قاعدة البيانات عن رصيد البنك وإسناده لمتغير نصي
            string MoneyOfBank = "سيتم جلبها من قاعدة البيانات";
            return MoneyOfBank;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using DevExpress.XtraEditors;

namespace DevExpressTest2
{
    public partial class frmExchangeMoney : XtraForm
    {
        public frmExchangeMoney()
        {
            InitializeComponent();
        }

        List<string> tresuries = new List<string>();

        private void frmExchange_Load(object sender, EventArgs e)
        {
            tresuries = GetTresuriesNames();
            comboBoxEdit1.Properties.Items.AddRange(tresuries);
            comboBoxEdit1.SelectedIndex = 0;
            comboBoxEdit2.Properties.Items.AddRange(tresuries);
            comboBoxEdit2.SelectedIndex = 1;
        }
```

```
        List<string> GetTresuriesNames()
        {
            // تحميل أسماء الخزنات من قاعدة البيانات .. وإعادتها كلائحة
            List<string> t = new List<string>() { "الافتراضية الخزينة",
                "الخزينة1",
                "الخزينة2" };
            return t;
        }

        private void comboBoxEdit1_SelectedIndexChangd(object sender, EventArgs e)
        {
            // يتم البحث في قاعدة البيانات عن الخزنة المحددة و عرض تفاصيلها
        }
    }
}
```

```
using DevExpress.XtraEditors;
using System.Collections.Generic;

namespace DevExpressTest2
{
    public partial class frmTresury : XtraForm
    {
        public frmTresury()
        {
            InitializeComponent();
        }

        List<string> tresuries = new List<string>();

        private void frmTresury_Load(object sender, System.EventArgs e)
        {
            tresuries = GetTresuriesNames();
            comboBoxEdit1.Properties.Items.AddRange(tresuries);
            comboBoxEdit1.SelectedIndex = 0;

            textEdit1.Text = GetTresuryAccount(comboBoxEdit1.Text);
        }

        string GetTresuryAccount(string t)
        {
            // يتم البحث في قاعدة البيانات عن رصيد الخزنة المحددة وإسناده لمتغير نصي
            string MoneyOfTresury = "البيانات قاعدة من جلبها سيتم";
            return MoneyOfTresury;
        }

        List<string> GetTresuriesNames()
        {
            // تحميل أسماء الخزنات من قاعدة البيانات .. وإعادتها كلائحة

            List<string> t = new List<string>() { "الافتراضية الخزينة",
                "الخزينة1",
                "الخزينة2" };
            return t;
        }
```

```
        private void comboBoxEdit1_SelectedIndexChangd(object sender, EventArgs e)
        {
            // يتم البحث في قاعدة البيانات عن الخزينة المحددة وعرض تفاصيلها
        }

        private void simpleButton1_Click(object sender, System.EventArgs e)
        {
            frmAddTresury fat = new frmAddTresury();
            fat.ShowDialog();
        }

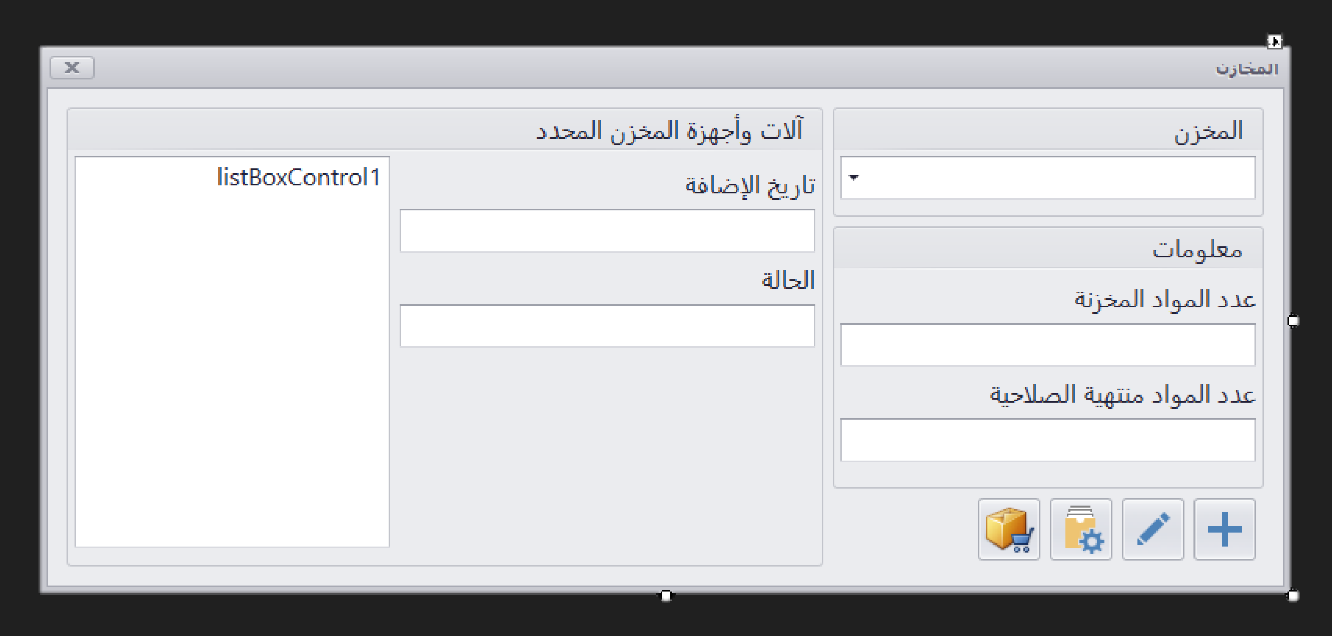
        private void simpleButton2_Click(object sender, System.EventArgs e)
        {
            frmBankAccount fba = new frmBankAccount();
            fba.ShowDialog();
        }

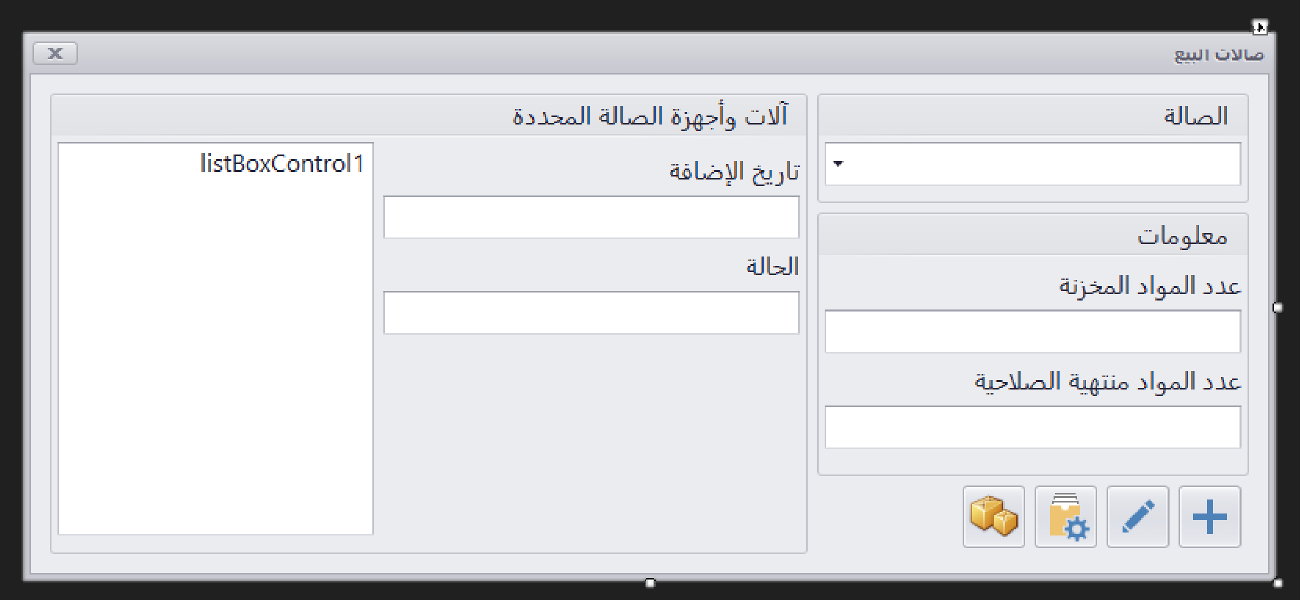
        private void simpleButton3_Click(object sender, System.EventArgs e)
        {
            frmExchangeMoney fe = new frmExchangeMoney();
            fe.ShowDialog();
        }

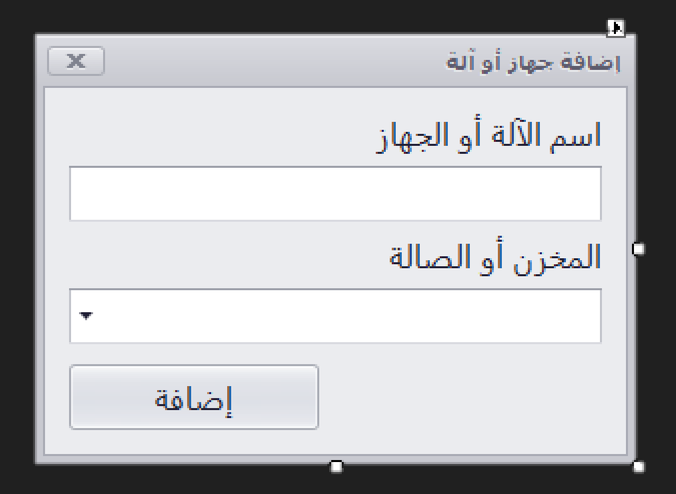
        private void simpleButton4_Click(object sender, System.EventArgs e)
        {
            frmAddMoney fam = new frmAddMoney();
            fam.ShowDialog();
        }
    }
}
```

نوافذ المخازن والصالات

قبل إدخال بيانات البضائع والمشتريات والمبيعات والزبائن ومصادر البضائع وغيرها، من المنطقي أن يتم ضبط المخازن والصالات أولًا.. وهذا ما سنصممه الآن. اعتمد على هذه التصاميم وطورها لخدمة المشروع أكثر:







الأدوات المستخدمة هي TextEdit وSimpleButton وComboBoxEdit وGroupControl وLabelEdit، ومضبوطة بشكل مشابه لما كان في الفقرة السابقة، فعد إليها.

استعن بالكود التالي للمخازن (وبشكل مشابه طور كود نافذة صالات البيع):

```
using System;
using System.Collections.Generic;
using DevExpress.XtraEditors;

namespace DevExpressTest2
{
    public partial class frmWarehouses : XtraForm
    {
        List<string> warehouses = new List<string>();

        public frmWarehouses()
        {
            InitializeComponent();
        }

        private void simpleButton1_Click(object sender, EventArgs e)
        {

            frmAddDevice fad = new frmAddDevice();
            fad.ShowDialog();
        }

        private void simpleButton4_Click(object sender, EventArgs e)
        {
            frmSalesHall fsh = new frmSalesHall();
            fsh.ShowDialog();
        }

        string GetWGoodsCount(string w)
        {
            // يتم البحث في قاعدة البيانات عن عدد مواد المخزن المحدد وإسناده لمتغير نصي
            string GoodsCount = "سيتم جلبها من قاعدة البيانات";
            return GoodsCount;
        }
```

```
        string GetWInvalidGoodsCount(string w)
        {
            // يتم البحث في قاعدة البيانات عن عدد مواد المخزن المحدد التالفة وإسناده لمتغير نصي
            string InvalidGoodsCount = "سيتم جلبها من قاعدة البيانات";
            return InvalidGoodsCount;
        }

        List<string> GetWDevicesNames()
        {
            // تحميل أسماء الخزنات من قاعدة البيانات .. وإعادتها كلائحة
            List<string> wd = new List<string>() { "جهاز1",
                "جهاز2",
                "جهاز3" };
            return wd;
        }

        List<string> GetWNames()
        {
            // تحميل أسماء الخزنات من قاعدة البيانات .. وإعادتها كلائحة
            List<string> w = new List<string>() { "الصالة الافتراضية",
                "الصالة1",
                "الصالة2" };
            return w;
        }

        private void frmWarehouses_Load(object sender, EventArgs e)
        {
            warehouses = GetWNames();
            comboBoxEdit1.Properties.Items.AddRange(warehouses);
            comboBoxEdit1.SelectedIndex = 0;

            textEdit1.Text = GetWGoodsCount(comboBoxEdit1.Text);
            textEdit2.Text = GetWInvalidGoodsCount(comboBoxEdit1.Text);
            listBoxControl1.Items.AddRange(GetWDevicesNames().ToArray());
        }
    }
}
```

أما بالنسبة لنافذة إضافة الأجهزة والآلات للصالات والمخازن:



```
using System;
using System.Collections.Generic;
using DevExpress.XtraEditors;

namespace DevExpressTest2
{
    public partial class frmAddDevice : XtraForm
    {
        public frmAddDevice()
        {
            InitializeComponent();
        }

        List<string> warehouses_saleshalls = new List<string>();
```

```
        private void frmAddDevice_Load(object sender, EventArgs e)
        {
            warehouses_saleshalls = GetWarehousesAndSalesHallsNames();
            comboBoxEdit1.Properties.Items.AddRange(warehouses_saleshalls);
            comboBoxEdit1.SelectedIndex = 0;
        }

        List<string> GetWarehousesAndSalesHallsNames()
        {
            // تحميل أسماء الخزنات من قاعدة البيانات .. وإعادتها كلائحة
            List<string> w_sh = new List<string>() { "الصالة الافتراضية",
                "الصالة1",
                "الصالة2",
                "المخزن الافتراضي",
                "المخزن1",
                "المخزن2", };
            return w_sh;
        }
    }
}
```

الملحقات

الملحق أ

قواعد بيانات SQLite

تتميز قواعد بيانات SQLite بصغر حجمها، وسرعتها، وسهولة استخدامها، وسهولة تضمينها مع البرامج.

كنا قد أسلفنا في كتابنا الأول أن قواعد بيانات SQL أوAccess تتطلب وجود مشغِّل لقاعدة البيانات مثبّت على جهاز المستخدم ليتمكن من تشغيل برامجك التي تحوي قواعد بيانات من هذين النوعين. قواعد بيانات SQLite لا تختلف عنهما في هذه النقطة، لكن ما تحتاجه ليس أكثر من مكتبة dll صغيرة تضعها ضمن ملفات المشروع.

قواعد بيانات هذا النوع موجهة للبرامج الصغيرة والمتوسطة، وهي مناسبة للمشاريع التعليمية بشكل كبير.

ما تحتاجه لإنشاء قواعد بيانات SQLite هو مشغّل قاعدة البيانات ويمكن تحميله من موقع SOURCEFORGE [1] أو من موقع SQLite [2]، ومدير قواعد البيانات SQLite من موقع SQLiteStudio [3].

ثم غيّر العنصر startup ضمن ملف app.config من Solution explorer ليصبح كالتالي:



```
<startup useLegacyV2RuntimeActivationPolicy="true">
  <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  <requiredRuntime version="v4.0.20506" />
</startup>
```

[1] تحميل SQLite من موقع SourceForge https://sourceforge.net/projects/sqlite-dotnet2/files/latest/download.
[2] تحميل SQLite من الموقع الرسمي https://system.data.sqlite.org/index.html/doc/trunk/www/index.wiki.
[3] تحميل SQLite Studio https://sqlitestudio.pl/index.rvt?act=download.

يمكنك إنشاء قواعد بيانات من خلال SQLiteStudio، وذلك بطريقة لا تختلف كثيرًا عن أنواع قواعد البيانات الأخرى، ثم ضع المكتبة sqlite3.dll بجوار البرنامج وأسندها إلى مراجعه، وبذلك تحظى بقاعدة بيانات سريعة وصغيرة الحجم (حجم SQLiteStudio لا يتعدى 30 ميغابايت وحجم sqlite3.dll حوالي 3 ميغابايت). ومع ذلك لا تستهون بها، فبإمكانها معالجة ما يفوق 500 إجراء في الثانية في تطبيقات الويب البسيطة! [1]

والكود التالي هو طبقة بيانات Data Layer يعطيك إمكانية الاتصال بقواعد بيانات SQLite والقيام بوظائف مخالفة (سنناقشها بعد الكود):



```
using System;
using System.Collections.Generic;
using System.Data;
using System.Windows.Forms;
using System.Data.SQLite;

namespace Eng27DB
{
    public class SQLiteDAL
    {
        #region Variables
        SQLiteConnectionStringBuilder cs = new SQLiteConnectionStringBuilder();
        SQLiteConnection connection;
        SQLiteCommand command;
        SQLiteDataAdapter adapter;
        SQLiteDataReader reader;
        DataTable data;
        #endregion


        #region Constructor
        public SQLiteDAL(string DataSource)
        {
            cs.DataSource = DataSource;
            cs.Version = 3;
            connection = new SQLiteConnection(cs.ConnectionString);
        }
        #endregion


        #region SettingUpConnection
        void OpenConnection()
        {
            try
            {
                if (connection.State != ConnectionState.Open)
                    connection.Open();
            }
```

[1] وادي التقنية – قواعد بيانات SQLite https://itwadi.com/what_is_SQLite.

```
            catch(SQLiteException ex)
            {
                MessageBox.Show("Could not connect to database" +
                    Environment.NewLine +
                    "More info: " + ex.Message);
            }
        }

        void CloseConnection()
        {
            try
            {
                if (connection.State != ConnectionState.Closed)
                    connection.Close();
            }
            catch (SQLiteException ex)
            {
                MessageBox.Show("Could not disconnect database" +
                    Environment.NewLine +
                    "More info: " + ex.Message);
            }
        }
        #endregion


        #region LoadData
        public DataTable LoadData(string TableName, bool IsDistinct)
        {
            OpenConnection();
            data = new DataTable();
            string selectword = IsDistinct ? "select distinct" : "select";
            string load = string.Format("{0} * from {1}" ,selectword, TableName);
            command = new SQLiteCommand(load, connection);
            adapter = new SQLiteDataAdapter(command);
            adapter.Fill(data); CloseConnection(); return data;
        }

        public DataTable LoadData(string TableName, string OnColumn,
            string OnValue, bool IsDistinct)
        {
            OpenConnection();
            data = new DataTable();
            string selectword = IsDistinct ? "select distinct" : "select";
            string load = string.Format
                ("{0} * from {1} where {2} = '{3}'",
                selectword, TableName, OnColumn, OnValue);
            command = new SQLiteCommand(load, connection);
            adapter = new SQLiteDataAdapter(command);
            adapter.Fill(data); CloseConnection(); return data;
        }

        public DataTable LoadData(string TableName,
            bool IsDistinct, params string[] ColumnNames)
        {
            OpenConnection();
            data = new DataTable();
            string columns = "";
            foreach (string s in ColumnNames)
                columns += s + ",";
            columns = columns.TrimEnd(',');
```

```
        string selectword = IsDistinct ? "select distinct" : "select";
        string load =
            string.Format("{0} {1} from {2}", selectword, columns, TableName);
        command = new SQLiteCommand(load, connection);
        adapter = new SQLiteDataAdapter(command);
        adapter.Fill(data); CloseConnection(); return data;
    }

    public DataTable LoadData(string TableName, string OnColumn,
        string OnValue, bool IsDistinct, params string[] ColumnNames)
    {
        OpenConnection();
        data = new DataTable();
        string columns = "";
        foreach (string s in ColumnNames)
            columns += s + ",";
        columns = columns.TrimEnd(',');
        string selectword = IsDistinct ? "select distinct" : "select";
        string load = string.Format
            ("{0} {1} from {2} where {3} = '{4}'",
            selectword, columns, TableName, OnColumn, OnValue);
        command = new SQLiteCommand(load, connection);
        adapter = new SQLiteDataAdapter(command);
        adapter.Fill(data); CloseConnection(); return data;
    }

    public string[] LoadColumn
        (string TableName, string ColumnName, bool IsDistinct)
    {
        List<string> column = new List<string>();
        OpenConnection();
        string selectword = IsDistinct ? "select distinct" : "select";
        string load = string.Format("{0} {1} from {2}",
            selectword, ColumnName, TableName);
        command = new SQLiteCommand(load, connection);
        reader = command.ExecuteReader();
        while (reader.Read())
        {
            column.Add(reader[0].ToString());
        }
        CloseConnection();
        return column.ToArray();
    }

    public string[] LoadColumn(string TableName, string ColumnName,
        string orderby, bool isDesc, bool IsDistinct)
    {
        List<string> column = new List<string>();
        OpenConnection();
        string load;
        string selectword = IsDistinct ? "select distinct" : "select";
        if (isDesc)
            load = string.Format("{0} {1} from {2} order by {3} desc",
                ColumnName, TableName, orderby);
        else
            load = string.Format("{0} {1} from {2} order by {3}", selectword,
                ColumnName, TableName, orderby);
        command = new SQLiteCommand(load, connection);
        reader = command.ExecuteReader();
        while (reader.Read())
```

```
            {
                column.Add(reader[0].ToString());
            }
            CloseConnection();
            return column.ToArray();
        }

        public string[] LoadColumn(string TableName, string ColumnName,
            string OnColumn, string OnValue, bool IsDistinct)
        {
            List<string> column = new List<string>();
            OpenConnection();
            string selectword = IsDistinct ? "select distinct" : "select";
            command = new SQLiteCommand(
                string.Format("{0} {1} from {2} where {3} = '{4}'", selectword,
                ColumnName,
                TableName,
                OnColumn,
                OnValue), connection);
            reader = command.ExecuteReader();
            while (reader.Read())
            {
                column.Add(reader[0].ToString());
            }
            CloseConnection();
            return column.ToArray();
        }
        #endregion
    }
}
```

الفئة السابقة تقوم بالترتيب بـ:

- الاتصال بقاعدة البيانات
- تحميل كل الأعمدة
- تحميل كل الأعمدة عند شرط ما
- تحميل أعمدة بعينها
- تحميل أعمدة بعينها عند شرط ما
- تحميل عمود معين
- تحميل عمود معين بترتيب عمود معين
- تحميل عمود معين عند شرط ما

طور هذه الفئة للوصول لبقية وظائف قواعد البيانات مثل إضافة وحذف وتعديل البيانات.

الملحق بـ

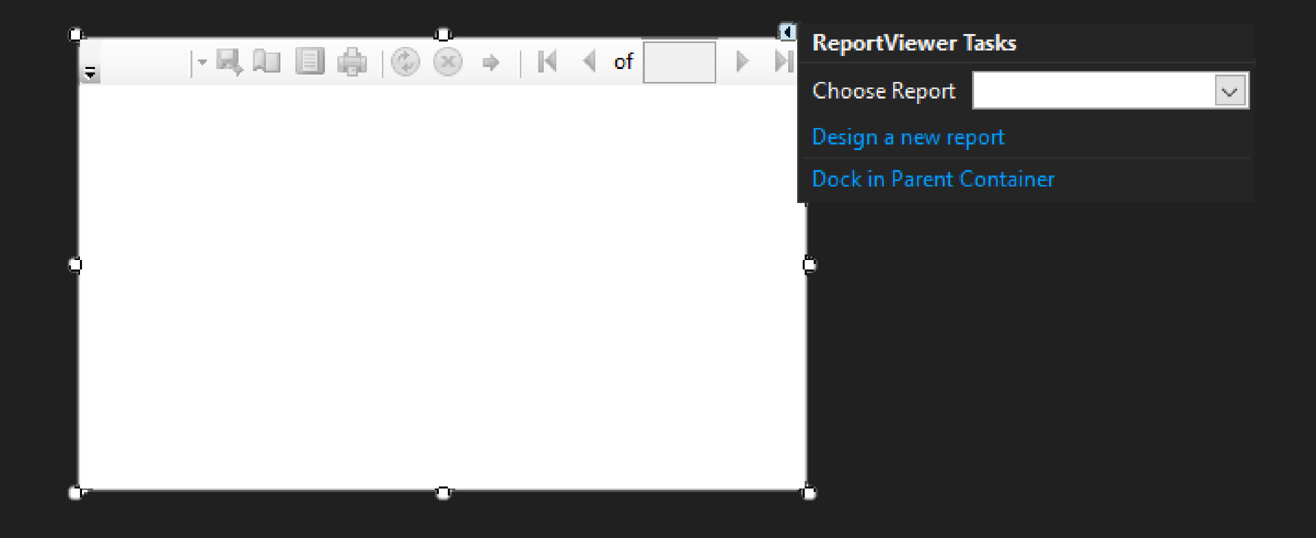
إنشاء التقارير لقواعد بيانات SQLite

إذا حاولت إنشاء تقرير لقاعدة بيانات SQLite ولم تستطع إيجاد مصدر البيانات Data Source لتزويد أدوات عرض التقارير Data Reporters؛ فهذا الملحق يناسبك، كما أنه يمكن الاعتماد عليه لقواعد البيانات الأخرى عند نفس المشكلة، ولكن على اعتبارها شائعة مع SQLite خصوصًا فقد عنونت الملحق بها.

سبب المشكلة أن مزوّد البيانات لم يتم تثبيته بشكل صحيح، مع أن برنامجك يتصل بقاعدة البيانات بلا أي مشاكل، فإذا لم تستطع حل المشكلة بإعادة تثبيت مزوّد البيانات فطبق الخطوات التالية.

ببساطة، ما سنقوم به، هو ما كان معالج التقارير Report Wizard يفترض أن يقوم به، وهو إيجاد مصدر البيانات (اعتمادًا على مزود البيانات)، واختيار الجداول المناسبة، وربط الحقول المطلوبة بالتقرير بعد تصميمه.

أنشئ أداة عرض تقارير (ولتكن Report Viewer):



إذا اخترت تصميم تقرير جديد Design a new report لن تجد مصدر البيانات، وفق المشكلة المدروسة في هذا الملحق:

النافذة الأولى (الموجودة خلف جميع النوافذ) هي معالج التقارير، ومنها نختار مصدر البيانات (النافذة الثانية)، والذي ننشئه باختيار مزود البيانات (النافذة الثالثة، الموجودة أمام النوافذ)، وكما هو واضح فلا يوجد عندي مزود بيانات SQLite.

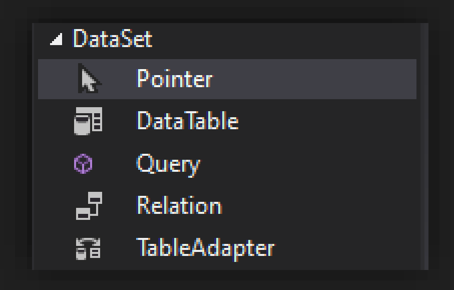
في الواقع، فإن ما يقوم به معالج التقارير هو اختيار مصدر البيانات ثم تصميم التقرير، وهو ما سنقوم به بأنفسنا وبلا منية حدا.

اخرج من النوافذ السابقة، وأنشئ مصدر بيانات جديد، وذلك من خلال متصفح المشروع Solution Explorer، بالنقر على اسم المشروع ثم Add New Item ثم البحث عن كائن من النوع DataSet:

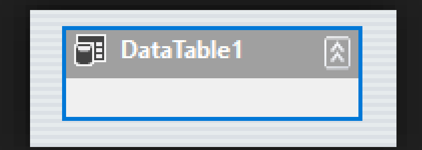
بشكل مشابه للمكوّنات Components، فإن الكائنات من النوع DataSet ستعرض لك هذه النافذة:

Use the Dataset Designer to visually create and edit typed datasets.
Drag database items from Server Explorer or the DataSet Toolbox onto the design surface, or right-click here to add new items.

وعليه، من صندوق الأدوات، أنشئ جدول بيانات، وذلك بإضافة كائن من النوع DataTable:

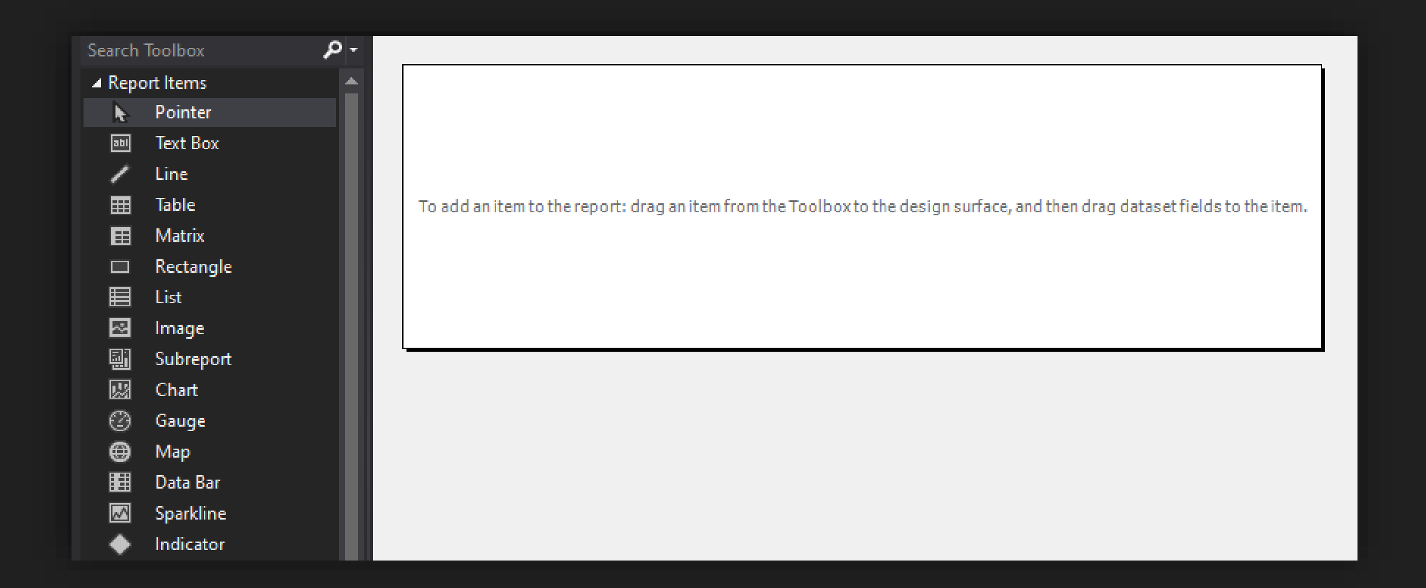


لتحصل على جدول فارغ:



انقر بالزر الأيمن على الجدول، ثم Add، ثم عمود Column، لتنشئ الأعمدة. غير أنواع البيانات التي ستحويها الأعمدة، وسمّ الجدول بالاسم الذي تريد.

الخطوة الثانية هي تصميم التقرير وربطه يدويًّا بمصدر البيانات الذي أنشأناه للتو. من مستعرض المشروع، اختر Add New Item ثم ابحث عن Report:



أضف جدولًا Table، لتظهر نافذة اختيار مصدر البيانات:

Dataset Properties

General

Choose the Dataset

Name:

DataSet1

Data source:

New...

Available datasets:

Fields:

Field Name	Type Name

Help

OK

Cancel

من Data source، اختر مصدر البيانات الذي أنشأته في الخطوة الأولى (يفترض أن يكون اسمه DataSet1. وفي الأسفل، اختر الجدول المراد عرضه.

وفي هذه الخطوة أيضًا، صمم التقرير باختيار أبعاده وما سيعرض فيه وغيرها من الخيارات.

الخطوة الثالثة، من خلال الكود، عند عرض التقرير؛ حمل البيانات وضعها في كائن من النوع DataTable، واربط هذا الكائن بمصدر البيانات الذي أنشأته في الخطوة الأولى (من المفترض أن اسمه DataSet1، إلا إذا قمت بتغييره أو أنشأت واحدًا قبله)، ثم اجعل التقرير الذي سيتم عرضه في الأداة ReportViewer مضمنًا فيها، بتمرير مساره ضمن المشروع (اسم مجال الأسماء، ثم اسم التقرير الذي أنشأته في الخطوة الثانية، بما في ذلك صيغته rdlc):



```
DataTable dt = db.LoadData("TABLENAME");
ReportDataSource rds = new ReportDataSource("DataSet1", dt);
reportViewer1.LocalReport.ReportEmbeddedResource = "YOURNAMESPACE.Report1.rdlc";

reportViewer1.LocalReport.DataSources.Clear();
reportViewer1.LocalReport.DataSources.Add(rds);
reportViewer1.RefreshReport();

this.reportViewer1.RefreshReport()
```

الملحق جـ

حماية قواعد بيانات SQLite

لا توجد طريقة مباشرة لقفل قواعد البيانات من نوع SQLite، وإنما تحتاج أن تقوم بذلك من خلال كود ما.

قبل القيام بأي عملية، أنشئ كلمة سر وذلك عند الاتصال بقاعدة البيانات:



```
SQLiteConnection conn =
    new SQLiteConnection("Data Source=MyDatabase.sqlite;Version=3;");
conn.SetPassword("password");
conn.Open();
```

في المرات القادمة عندما تتصل بقاعدة البيانات، مرر كلمة السر مع جملة الاتصال:



```
conn = new SQLiteConnection
    ("Data Source=MyDatabase.sqlite;Version=3;Password=password;");
conn.Open();
```

وإذا رغبت بتغيير كلمة السر:



```
conn.ChangePassword("new_password");
```

ولإزالتها:



```
conn.ChangePassword(String.Empty);
```

الملحق د

الفئات الوسيطة، نقل البيانات بين النوافذ، الإعدادات

يمكن الوصول لمتغيرات نافذة Form ما من نافذة أخرى بجعلها عامة public (أو بشكل عام: الوصول لأعضاء فئة ما من فئة أخرى)، ولكن هذا الأسلوب يجبرك على استنساخ النوافذ (الفئات) قبل الوصول لأعضاءها، أي أنه لا يمكنك الوصول للمتغيرات إلا بعد استنساخ النوافذ وليس قبلها.

يمكنك اعتماد أسلوب آخر، أنشئ فئة ضع فيها كل ما يتعلق ببرنامجك، من إعدادات ومعلومات عامة وغيرها، وما ستقوم به هذه الفئة هي كتابة الإعدادات والخيارات في مصدر بيانات ما (ملف أو قاعدة بيانات أو رجستري أو غيرها) والقراءة منها. هذه الفئة يجب أن تُستنسخ في جميع نوافذ وفئات البرنامج والتي تحتاج البيانات العامة.

هذه الفئة الوسيطة – ولتكن باسم Settings – يجب أن تحوي إجراءات لتحميل البيانات وإجراءات لحفظها، وإجراءات لحفظ إعداد بعينه وأخرى لقراءتها. كما يجب أن تحوي خصائص تمثل الإعدادات، بحيث يتم الوصول لإعداد معين من الإعدادات من خلال الوصول لخاصية معينة. الملحق هـ كائنات البيانات الخاصة فيه المزيد، فليُراجَع.

يمكنك إما تمرير هذه الفئة كوسيط للتابع البناء لنوافذ البرنامج، وعندها ستكون جميع النوافذ تصل لنفس الإعدادات، أو إنشاء خاصية في جميع النوافذ من النوع Settings (الفئة الوسيطة التي نناقشها)، وإسناد كائن من هذا النوع عند استنساخ هذه النوافذ للمرة الأولى، وفي هذه الحالة أيضًا ستحصل جميع النوافذ على نفس الإعدادات.

الملحق هـ

كائنات البيانات الخاصة

عند قراءة بيانات من قواعد البيانات (أو أي مصدر للبيانات) فإنك قد تحتاج لتخزينها للتعامل معها لاحقًا، كإجراء الحسابات والإحصائيات عليها. إذا أردت تخزين بيانات عمود معين يمكنك ذلك من خلال المتغيرات التقليدية (العددية int وdouble، والنصية string، وغيرها)، بإنشاء مصفوفة منها. أما إذا أردت تخزين مجموعة من الأعمدة، فعليك إنشاء عدة مصفوفات، وهذا فضلًا عن أنه مربك، فهو أسلوب غير متقَن.

الفكرة ببساطة هي إنشاء فئة فيها خصائص، تمثل الأعمدة المراد تخزين بيانات، ثم إنشاء مصفوفة من كائنات هذه الفئة. الأداة ChartPie من الفصل السابع فيها هذه الفكرة. حتى أن بعض أدوات عرض البيانات تتعامل مع المصفوفات كمصدر للبيانات.

فهرس المصطلحات Glossary (عربي)

أ

أداة Control

الأدوات في مشاريع النوافذ هي مكونات تحوي وظائف مختلفة تستخدم في تطبيقات ويندوز من جهة المستخدم.

أداة مستخدم User Control

أداة محتوى، والتي يمكنها أن تحوي كائنًا ما أو أكثر من أي نوع من الأدوات (مثل النصوص أو الصور أو اللوائح).



بايت Byte

وحدة القياس الأساسية في الحواسيب.

بِت Bit

أصغر وحدة قياس في الحاسوب.

البرمجة كائنية التوجه OOP

أسلوب برمجي يقوم على أربعة مبادئ، بحيث يتم التعامل مع أجزاء البرنامج على أنها أشياء أو كائنات.

برنامج Program

مجموعة من التعليمات، أو الوحدات أو التوابع، التي تسمح بالقيام بعملية معينة في الحاسوب.

بنية Block

قسم من كود أو خوارزمية في برنامج.

البيانات Data

معلومات تم معالجتها أو تخزينها من قبل الحاسوب.



تابع Function

طريقة تعيد قيمة ما.

تجربة المستخدم UX

سلوك المستخدم حول استخدامه لمنتج ما، أو خدمة أو نظام.

تطبيق Application

برنامج أو مجموعة من البرامج، مصمم للمستخدم النهائي.

تعدد الأشياء Polymorphism

قدرة – أو قابلية – الكائن على أخذ أكثر من شكل (أو التواجد في أكثر من شكل).

تعليق Comment

شرح أو ملاحظة أو حاشية للكود، لا يُترجَم.

التغليف Encapsulation

تجميع البيانات، مع العمليات التي تجرى عليها، في بنية واحدة.

التوثيق XML Documentation

تعليقات نصية تستخدم لوصف الفئات وأعضاء الفئات للمبرمجين.



جملة الاختيار Switch Statement

آلية تسمح لقيمة متغير ما أن تتحكم بتغيير مجريات تنفيذ البرنامج.

جملة شرط إذا If Statement

بنية منطقية تستخدم في البنى البرمجية.

جهاز إدارة ويندوز WMI

تقنية (أو أداة إدارية) من ميكروسوفت، والتي تقدم معلومات عن أجهزة نظام التشغيل وتجري عمليات عليها.



حدث Event

إشعار يُرسل من قبل كائن ما للإعلام بحدوث حدث ما.

حرف char

نوع بيانات يستخدم لتخزين حرف واحد بالترميز Unicode.



خاصية Property

عضو من أعضاء الفئات والتي تزودها بآلية لقراءة وكتابة وحساب قيمة حقل خاص لأحد الكائنات.

خدمة API

الترجمة الحرفية لهذا المصطلح واجهة برمجية لتطبيق ما، وهي خدمة يقدمها طرف أول لطرف ثان، دون تقديم تفاصيل هذه الخدمة أو مبدأ عملها (وإلا لأصبحت شيفرة مصدرة Source Code)، بحيث يستفيد الطرف الثاني منها في منتجاته بقيود معينة.

خدمة ويندوز Windows Service

برنامج ويندوز يعمل في الخلفية، لفترة طويلة.

خطأ Error

سطر برمجي غير مدروس يؤدي إلى تشغيل البرنامج بشكل غير صحيح، مما يجعله ينهار.



دالة Function

راجع: تابع Function.



شرط Condition

ميزة أساسيات من مزايا لغات البرمجة، التعابير الشرطية هي تعابير يتم تنفيذها في إحدى حالتين متضادتين.

شريط الوصول السريع QAT

شريط يستخدم في مشاريع الأشرطة Ribbons لاحتواء الأوامر والاختصارات الأكثر شيوعا في التطبيق.



صندوق الأدوات Toolbox

نافذة أو لائحة تحوي أيقونات وأزرار تمثل أدوات برمجية.

صيغة Syntax

مجموعة من القواعد التي تعرّف الروابط بين الرموز والكلمات المحجوزة في لغة ما؛ حتى تكون صحيحة بنيويًّا.

ع

عدد صحيح int

نوع بيانات يستخدم لتخزين القيم الرقمية التي لا تحوي فاصلة عشرية أو كسور.

عدد عشري double

نوع بيانات يستخدم لتخزين الأعداد العشرية.

علامة مائية Watermark

رسالة (لوغو أو توقيع) توضع على صورة أو ورقة أو وسيلة إدخال، بشفافية عالية.

ط

طريقة Method

مجموعة من الأسطر البرمجية التي تؤدي وظيفة معينة.

طور التصميم Design-Time

عملية إنشاء تطبيق ما، وتصميم واجهاته، وضبط خصائصه، وكتابة الكود.

طور التنفيذ Run-Time

الفترة التي يكون فيها البرنامج قيد العمل.

ف

فئة Class

نوع بيانات مرجعي، أصل الكائنات.

ق

قاعدة البيانات Database

آلية تخزين البيانات ومعالجتها.



كائن Object

نسخة من فئة ما. في الواقع، فإن كل شيء في البرمجة بلغة #C عبارة عن كائن.

الكود النظيف Clean Code

كود سهل القراءة والفهم، والتعديل.



مبدأ المسؤولية الواحدة SRP

مبدأ ينص على أن كل وحدة برمجية أو فئة أو طريقة ينبغي أن تكون مسؤولة عن جزء وحيد من وظائف البرنامج، والتي يجب أن يتم تغليفها.

المترجم Compiler

برنامج حاسوب يترجم شيفرة مصدرية (كود) مكتوبة بلغة من اللغات العليا (مثل ++C) ويحولها لتعليمات لغة الآلة، والتي يمكن للمعالج أن يفهمها.

متغير Variable

موقع تخزيني من الذاكرة يستخدم لتخزين البيانات في وقت التنفيذ.

مشروع Project

ورشة حاسوبية لإنشاء برنامج.

مشروع تطبيق سطري Windows Console Application Project

مشروع يمكن من خلاله إنشاء تطبيقات سطرية.

مشروع تطبيق نوافذ WinForms Projects

مشروع يمكن من خلاله إنشاء تطبيقات رسومية.

مشروع مكتبة أدوات Windows Control Library Project

مجموعة من الفئات التي تمثل أدوات تستخدم في نماذج ويندوز، والتي يمكن تخصيصها واستخدامها من قبل المبرمجين لإنشاء البرامج والتطبيقات.

مشروع مكتبة فئات Windows Class Library Project

مجموعة من الفئات أو النماذج المبرمجة المكتوبة مع بعضها في ملف ما (عادة ملف DLL)، والتي يمكن تخصيصها واستخدامها من قبل المبرمجين لإنشاء البرامج والتطبيقات.

مصفوفة Array

مجموعة من المتغيرات المستخدمة لتخزين بيانات من نفس النوع في وقت التنفيذ.

معالمات Arguments

قيمة يتم تمريرها بين البرامج أو التوابع أو الإجراءات.

مكوّن Component

فئة توظف أعضاء الواجهة IComponent (ترث منها). لاحظ أن هذه الفئات لا تحوي واجهة رسومية.

ملف File

كائن في الحاسوب يستخدم لتخزين البيانات والمعلومات والإعدادات والأوامر ليتم استخدامها من قبل برنامج ما.

ملف DLL

الملفات من النوع DLL تدعى ملفات الارتباط الحيوي، هي عبارة عن مكتبات دعم برامج أنظمة التشغيل.

ملف EXE

الملفات التنفيذية في أنظمة ويندوز.

ملف APK

الملفات التنفيذية في أنظمة أندرويد.

مواصفة Attribute

ميزة من مزايا لغة C#، والتي تضيف معلومات لأجزاء المشروع، وكائناته المختلفة.



نص string

نوع بيانات يستخدم لتخزين سلسلة من المحارف.

و

واجهة المستخدم UI

النسق الرسومي لمنتج ما.

الوراثة Inheritance

أحد مبادئ البرمجة كائنية التوجه. هو آلية نسخ أعضاء فئة ما لفئة جديدة.

وسيط Parameter

قيمة يتم تمريرها للتوابع أو الإجراءات.

فهرس المصطلحات Glossary (English)

A

API

An application programming interface (API) is a computing interface that defines interactions between multiple software intermediaries.

APK Files

An Android standard package files.

Application تطبيق

A program or group of programs designed for end users.

Arguments معاملات

A value that is passed between programs, subroutines or functions.

Array مصفوفة

A set of Variables used to store data of the same type at Run-Time.

Attribute مواصفة

A powerful feature in the C# programming language that can add metadata information to your assemblies. An attribute is actually an object that is associated with any of these elements: Assembly, Class, Method, Delegate, Enum, Event, Field, Interface, Property and Struct.

B

Bit بِت

A bit (short for binary digit) is the smallest unit of data in a computer.

Block بنية

A block is a section of software code or an algorithm in software programming.

Byte بايت

The basic unit of information in computer storage and processing.

C

Character (char) حرف

A data type used to hold a single, unicode character.

Class فئة

A reference type, and a blueprint for creating objects.

Clean Code الكود النظيف

A code that is easy to read, understand and change.

Comment تعليق

a programmer-readable explanation or annotation in the source code of a computer program, which is not compiled.

Compiler المترجم

A computer software that translates (compiles) source code written in a high-level language (e.g., C++) into a set of machine-language instructions that can be understood by a digital computer's CPU.

Component مكوّن

A class that implements the IComponent interface, which indicates that a class can interact with it's logical container. Note that these classes don't have GUI.

Condition شرط

Main feature of programming languages, conditional expressions are expressions that evaluate to either true or false.

Control أداة

Controls in WinForms are reusable components that encapsulate user interface functionality and are used in client-side Windows applications.



Data بيانات

Information processed or stored by computer.

Database قاعدة بيانات

A mechanism of storing, operating and processing data.

Design-Time طور التصميم

The process of creating an application, designing an interface, setting the properties, and writing the code.

DLL Files

Files from type DLL (Dynamic Link Libraries) are libraries to support OS programs.

Double (double) عدد عشري

A data type used to hold numbers with decimal points.



Encapsulation التغليف

The bundling of data, along with the methods that operate on that data, into a single unit.

Error خطأ

A bug in a program that causes it to operate incorrectly, but not to terminate abnormally (or crash).

Event حدث

A notification sent by an object to signal the occurrence of an action.

EXE Files

Executable files in Windows OS.

F

File ملف

An object on a computer that stores data, information, settings, or commands used with a computer program.

Function تابع

A method, that returns a value.

I

If Statement جملة شرط إذا

Logical block used within programming block.

Inheritance الوراثة

One of the concepts of OOP, it is the mechanism of acquiring class members to a new class.

Integer (int) عدد صحيح

A data type used to hold numeric values that contains only whole numbers and cannot contain fractions.

M

Method طريقة

A code block that contains a series of statements.

O

Object كائن

An instance of a class. Actually, everything in C# programming language is an object.

OOP Objected Oriented Programming البرمجة كائنية التوجه

OOP is a programming which is based on four techniques, whereby parts of programs are treated as Objects or Things.

P

Parameter وسيط

a value that is passed into functions.

Polymorphism تعدد الأشكال

The ability of an object to take on many forms.

Program برنامج

A set of instructions, or a set of modules or procedures, that allow for a certain type of computer operation.

Project مشروع

A computer workshop to for creating a program.

Property خاصية

A member that provides a flexible mechanism to read, write, or compute the value of a private field in an instance of class.

Q

QAT Quick Access Toolbar شريط الوصول السريع

Toolbar used in Ribbon Forms to contain common commands and shortcuts to the application.

R

Run-Time طور التنفيذ

Runtime is the period of time when a program is running.

S

SRP Single Responsibility Principle مبدأ المسؤولية الواحدة

A principle that states that every module, class or function in a computer program should have responsibility over a single part of that program's functionality, which it should encapsulate.

String (string) نص

A data type used to hold series of characters that is used to represent text.

Switch Statement جملة الاختيار

A type of selection control mechanism used to allow the value of a variable or expression to change the control flow of program execution via search and map.

Syntax صيغة

The set of rules that defines the combinations of symbols that are considered to be correctly structured.

T

Toolbox صندوق الأدوات

A window or pane that contains icons and buttons that are tools in the program.

U

UI User Interface واجهة المستخدم

The graphical layout of an application.

User Control أداة مستخدم

A ContentControl, which means that it can contain a single object of any type (such as a string, an image, or a panel).

UX User Experience تجربة المستخدم

A person's emotions and attitudes about using a particular product, system or service.



Variable متغير

A memory storage location used to store values at Run-Time.



Watermark علامة مائية

A message (usually a logo, stamp, or signature) superimposed onto an image, paper or input box, with a great deal of transparency.

Windows Service خدمة ويندوز

Standard Windows program that operate in background, for a long time.

Windows Class Library Project مشروع مكتبة فئات

A collection of prewritten classes or coded templates, which can be specified and used by a programmer when developing an application program.

Windows Console Application Project مشروع تطبيق سطري

A program designed to be used via a text-only computer interface.

Windows Control Library Project مشروع مكتبة أدوات

A collection of classes that represent WinForms controls, which can be specified and used by a programmer when developing an application program.

Windows Forms Application Project (WinForms Project) مشروع تطبيق نوافذ

A graphical user interface application programming interface (API) included as a part of Microsoft's .NET Framework.

WMI Windows Management Instrumention جهاز إدارة ويندوز

A technology (or administrative tool) by Microsoft, that provides information on OS devices and performs operations on them.



XML Documentation التوثيق

Text tags used to describe classes and its members to programmers.

إلى هنا، تنتهي رحلتك معي في عالم البرمجة والتصميم بلغة #C، على امتداد اثني عشر فصلًا، موزعة على أربعة أبواب، إذ بدأنا ببعض المفاهيم الشائعة في عالم البرمجة، وانتقلنا لأساسيات التصميم، ثم طبقنا هذه المفاهيم والأساسيات في مشاريع حقيقية لتصميم الأدوات والنوافذ. فأسأل الله أن ينفعك وإياي بها، وأن يجعلها حجة لنا لا علينا.

لا يوجد قائمة بمراجع محددة لهذا الكتاب، فأغلب محتويات الكتاب لا تستند لمرجع معتمد، في حين أن بعضه نقلته من مراجع معينة وأشرت لذلك في حواشي صفحات الكتاب. لكن مع ذلك، يمكنك الرجوع إلى مدونتي للحصول على قوائم بالمصادر والمراجع التي أعتمد عليها عمومًا.

وختامًا، وكأي عمل في هذا الدنيا، فإنه لم يكن ليكتمل لولا فضلٌ من الله وتوفيقه، فله الحمد والمنة! وآخر دعوانا أن الحمد لله رب العالمين.

حسن الفحل، 2020/12/13